IMPLEMENTACIÓN DE REDES NEURONALES ARTIFICIALES EN HARDWARE RECONFIGURABLE.

FAUSTO MIGUEL CASTRO CAICEDO
RIGO NELSON BENAVIDES CERÓN

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
SAN JUAN DE PASTO
2010

IMPLEMENTACIÓN DE REDES NEURONALES ARTIFICIALES EN HARDWARE RECONFIGURABLE.

FAUSTO MIGUEL CASTRO CAICEDO
RIGO NELSON BENAVIDES CERÓN

Trabajo de grado presentado como requisito parcial para optar el título de Ingeniero Electrónico

Director del proyecto:
Mg. JAIME ORLANDO RUIZ PAZOS
Ingeniero Electrónico

UNIVERSIDAD DE NARIÑO
FACULTAD DE INGENIERÍA
PROGRAMA DE INGENIERÍA ELECTRÓNICA
SAN JUAN DE PASTO
2010

“Las ideas y conclusiones aportadas en la tesis de grado, son responsabilidad exclusiva de los autores.”

Artículo 1º. Acuerdo No. 324 de Octubre 11 de 1966, emanado del Honorable Consejo Directivo de la Universidad de Nariño

Nota de Aceptación:

Firma del Presidente del Jurado

Firma del Jurado

Firma del Jurado

San Juan de Pasto, Febrero de 2010.

Dedicamos:

A nuestros Padres que nos dieron la vida y han estado con nosotros en todo momento. Gracias por todo, por darnos una carrera para nuestro futuro y por creer en nosotros, aunque hemos pasado momentos difíciles siempre han estado apoyándonos y brindándonos todo su amor.
Los queremos con todo el corazón y este trabajo, que nos llevó tiempo en culminar, es para ustedes, por ser los mejores padres del mundo.

A nuestros hermanos, sobrinos y seres queridos por estar con nosotros brindándonos todo su amor y apoyo incondicional.

A toda nuestra familia, quienes han sido un ejemplo de superación para poder alcanzar nuestras metas.

El verdadero sabio sólo es riguroso consigo mismo; con los demás es amable.
PLUTARCO

AGRADECIMIENTOS

Expresamos nuestras más sinceras muestras de agradecimiento a:

Dios, por enseñarnos el camino correcto de la vida, guiándonos y fortaleciendo cada día nuestro espíritu.

Ing. Jaime Orlando Ruiz Pazos, director del trabajo de grado, por facilitarnos la principal herramienta para culminar este trabajo y por su permanente disposición desinteresada a la hora de guiarnos en distintas fases del proyecto.

Nuestros padres, Astheria Cerón, Rosalina Caicedo, Jonás Benavides, Rodrigo Castro, por su paciencia y por creer siempre en nosotros, apoyándonos en todas las etapas de nuestras vidas.

La Universidad de Nariño, por brindarnos la oportunidad de culminar nuestra carrera, que nos permite tener un futuro mejor.

Grupo de docentes del programa de Ingeniería Electrónica de la Universidad de Nariño, por sus consejos y por compartir con nosotros sus amplios conocimientos y experiencias.

Nuestros compañeros de estudio, por el apoyo que de ellos hemos recibido.

Los Autores.

RESUMEN

El propósito central del presente trabajo es la integración de las tecnologías FPGA y las Redes Neuronales Artificiales y surge como una alternativa para la creación de sistemas electrónicos complejos que se espera, contribuyan a resolver importantes problemas tecnológicos; tras esta perspectiva, se comienza por adquirir los componentes tanto teóricos como experimentales que permiten estimar y usar de manera eficiente los recursos de las tecnologías reconfigurables a la hora de implementar redes neuronales. Posteriormente se analiza un caso de estudio, el cual consiste en entrenar una red neuronal en software que determine cuando un tumor mamario es benigno o maligno. Finalmente, a partir de los conceptos adquiridos y los resultados logrados, la red neuronal obtenida se implementa en un FPGA Spartan 3 de la familia Xilinx haciendo uso del lenguaje de descripción de hardware VHDL.

Palabras Claves: Redes Neuronales, FPGA, VHDL.

ABSTRACT

The central aim of the present research is the integration of the technologies FPGA and the Artificial Neural Networks and it arises like an alternative for the creation of electronic complex systems and it's expected they contribute to solve important technological problems; according to this, firstly it`s necessary to acquire the components so much theoretical as experimental that allow to estimate and to use in an efficient way the resources of configurable technologies when implementing neural networks. Later on a case of study is analyzed, which consists on training a neural network in software to determine when a breast tumor is benign or malicious. Finally, the neuronal network obtained is implemented in a Spartan 3 FPGA of the family Xilinx making use of the hardware description language VHDL.

Key Words: Neuronal Networks, FPGA, VHDL.

CONTENIDO

Pág.

INTRODUCCIÓN 21

1. MARCO CONCEPTUAL 22

1.1 REDES NEURONALES ARTIFICIALES (RNA) 22

1.1.1 Estructura de un sistema neuronal artificial (ANS) 23

1.1.2 Modos de operación de una red neuronal 24

1.1.2.1 Modo de entrenamiento 24

1.1.3 Capacidad de generalización 25

1.1.4 Patrones de entrenamiento 26

1.1.4.1 Modo incremental 26

1.1.4.2 Modo batch 26

1.1.5 Clasificación de las redes neuronales artificiales 26

1.1.5.1 Perceptrón 27

1.1.5.2 Adalina / Madalina 28

1.1.5.3 Perceptrón multicapa (MLP) 29

1.1.6 Algoritmo backpropagation 31

1.2 LÓGICA RECONFIGURABLE 31

1.2.1 Lenguajes de descripción de hardware (HDLs) 32

1.2.2 Codiseño hardware / software 33

1.2.3 Dispositivos lógicos programables (PLDs) 33

1.3 FPGA (FIELD PROGRAMMABLE GATE ARRAY) 34

1.3.1 Programación de los FPGAs 36

1.3.2 Evolución de los FPGAs 36

1.3.2.1 FPGAs de Xilinx 37

2. ETAPA INVESTIGATIVA 39

2.1 DEFINICIÓN DEL MODELO MATEMÁTICO DE UNA RED NEURONAL 39

2.1.1 Representación matricial de una neurona estándar 39

2.1.2 Representación matricial de una capa de neuronas 39

2.1.3 Representación matricial de redes multicapa 40

2.2 PARALELISMO EN REDES NEURONALES 42

2.2.1 Paralelismo de capa 42

2.2.2 Paralelismo de nodo 42

2.2.3 Paralelismo de sinapsis 42

2.2.4 Paralelismo sistémico 43

2.3 ARITMÉTICA EN REDES NEURONALES 43

2.3.1 Representación de datos 43

2.3.1.1 Mínimo rango de precisión permitido 44

2.3.2 Procesamiento de suma de productos 44

2.3.3 Funciones de activación 45

2.3.3.1 Método por manejo de tablas 45

2.3.3.2 Método por aproximaciones polinómicas 47

2.4 MODOS DE ENTRENAMIENTO DE UNA RED NEURONAL EN FPGA 48

2.4.1 Entrenamiento externo (offline learning) 48

2.4.2 Entrenamiento interno (on-chip learning) 48

2.5 BLOQUES BÁSICOS DE UNA RED NEURONAL 48

2.5.1 Módulos de entrada 49

2.5.2 Unidad de cómputo paralelo 49

2.5.2.1 Unidad de procesamiento PE 49

2.5.2.2 Módulo para la función de activación 49

2.5.3 Interfaz 49

2.5.4 Bloque algorítmico 50

2.5.5 Sistema de comunicaciones 50

2.5.5.1 Arquitectura orientada a bus o BBA (broadcast bus architecture) 50

2.5.5.2 Arquitecturas sistólicas 51

2.5.6 Unidad de almacenamiento 53

2.5.6.1 Memoria de almacenamiento distribuida 53

2.5.6.2 Memoria de almacenamiento centralizada 53

2.5.7 Módulos de salida 53

2.5.8 Unidad de control 54

2.5.8.1 SISD (single instruction, single data) 54

2.5.8.2 SIMD (single instruction, multiple data) 54

2.5.8.3 MISD (multiple instruction, single data) 54

2.5.8.4 MIMD (multiple instruction, multiple data) 54

3. ETAPA DE DISEÑO SOFTWARE 55

3.1 ESTUDIO Y PLANTEAMIENTO DE LA APLICACIÓN 55

3.2 DISEÑO DE LA RED NEURONAL 56

3.3 ENTRENAMIENTO SOFTWARE DE LA RED NEURONAL 58

3.4 ANÁLISIS DE RESULTADOS DE LA ETAPA DE DISEÑO SOFTWARE 60

4. ETAPA DE DISEÑO HARDWARE 63

4.1 DISEÑO VHDL 63

4.1.1 Prototipos hardware 64

4.1.1.1 Prototipos combinacionales 64

4.1.1.2 Prototipos secuenciales 67

4.2 CARACTERÍSTICAS DEL SISTEMA 72

4.3 DISEÑO DEL SISTEMA NEURONAL ARTIFICIAL 73

4.4 CONFORMACIÓN DE BLOQUES FUNCIONALES 74

4.4.1 Interfaz y módulo de entrada 74

4.4.2 Unidad de cómputo paralelo 75

4.4.2.1 Funcionamiento interno de un procesador elemental (PE) 76

4.4.2.2 Lógica de la función de activación 78

4.4.2.3 Funcionamiento de la unidad de cómputo paralelo 79

4.4.2.4 Aspectos generales de la representación de datos 80

4.4.3 Sistema de comunicaciones 81

4.4.4 Unidad de almacenamiento 81

4.4.5 Unidad de control 82

4.4.6 Módulo de salida 85

4.5 ANÁLISIS DE RESULTADOS DE LA ETAPA DE DISEÑO HARDWARE 86

4.5.1 Análisis del desempeño de una red neuronal desarrollada en software con respecto a otra implementada en hardware reconfigurable 87

APORTE A FUTUROS TRABAJOS 89

CONCLUSIONES 90

BIBLIOGRAFÍA 92

NETGRAFÍA 94

ANEXOS 95

LISTA DE TABLAS

Pág.

Tabla 1. Funciones de activación habituales. 23

Tabla 2. Familias de FPGAs de Xilinx. 38

Tabla 3. Clasificación según las salidas de la red. 57

Tabla 4. Almacenamiento de los datos en la memoria ram de un PE. 78

Tabla 5. Instrucciones de control estáticas. 83

Tabla 6. Instrucciones de control dinámicas. 83

Tabla 7. Reporte final del proceso de síntesis. 86

Tabla 8. Comparación hardware reconfigurable frente a software. 88

Tabla 9. Características de la familia spartan 3. 98

LISTA DE FIGURAS

Pág.

Figura 1. Modelo de neurona estándar. 23

Figura 2. Estructura jerárquica de un sistema basado en RNA. 24

Figura 3. Clasificación de las redes neuronales artificiales. 27

Figura 4. Arquitectura del perceptrón. 27

Figura 5. Arquitectura de una red adalina. 29

Figura 6. Regiones de decisión del MLP. 30

Figura 7. Arquitectura de un MLP de dos capas. 31

Figura 8. Estructura básica de los FPGA. 35

Figura 9. Modelo matricial de una neurona estándar. 39

Figura 10. Modelo matricial de una capa de neuronas. 40

Figura 11. Modelo matricial de redes multicapa. 41

Figura 12. Sistemas de representación de datos. 43

Figura 13. Configuraciones habituales para suma de productos. 45

Figura 14. Arquitectura BBA. 51

Figura 15. Representación del Flujo de Datos. 51

Figura 16. Arquitectura sistólica SRA. 52

Figura 17. Arquitectura sistólica SRAGB. 52

Figura 19. Arquitectura de red. 57

Figura 20. Ventana para cargas datos de entrenamiento. 58

Figura 21. Selección del número de neuronas de la capa oculta. 59

Figura 22. Ventana para entrenar la red neuronal. 59

Figura 23. Progreso del entrenamiento de la RNA. 61

Figura 24. Evolución del error para los 3 conjuntos de datos. 61

Figura 25. Gráfica de la matriz de confusión. 62

Figura 26. Símbolo y Tabla de Verdad de la Puerta AND y OR. 64

Figura 27. Símbolo del sumador con revisión de overflow. 65

Figura 28. Símbolo multiplicador paralelo. 65

Figura 29. Símbolo y tabla del mux de dos entradas de datos. 66

Figura 30. Símbolo y tabla de comportamiento para un decodificador binario de 4 bits. 67

Figura 31. Registro de n bits con carga paralela y clear asíncrono. 68

Figura 32. Contador ascendente. 68

Figura 33. Contador ascendente con carga paralela. 69

Figura 34. Ram de único y doble puerto. 69

Figura 35. Diagrama del registro rotativo de carga múltiple. 70

Figura 36. Símbolo circuito UART. 71

Figura 37. Símbolo circuito organizador. 72

Figura 38. Diagrama del ANS implementado. 73

Figura 39. Interfaz y módulo de entrada. 75

Figura 40. Unidad de cómputo paralelo. 76

Figura 41. Procesador elemental. 77

Figura 42. Función de activación. 78

Figura 43. Unidad de almacenamiento de datos intermedios. 81

Figura 44. Diagrama de flujo de la unidad de control. 84

Figura 45. Visualización de resultados. 86

Figura 46. Tarjeta de desarrollo S3. 96

Figura 47. Componentes de la tarjeta de desarrollo S3. 97

Figura 48. Arquitectura de la spartan 3. 99

Figura 49. Ventana principal Ise 7.1 102

Figura 50. Ventana para crear nuevo proyecto. 102

Figura 51. Propiedades del dispositivo. 103

Figura 52. Creación de un nuevo módulo VHDL. 104

Figura 53. Definición de entradas y salidas. 104

Figura 54. Síntesis del diseño. 105

LISTA DE ANEXOS

Pág.

ANEXO A. PLATAFORMA DE DESARROLLO HARDWARE SPARTAN 3 96

ANEXO B. FLUJO DE DISEÑO PARA FPGAs 99

ANEXO C. ISE WEB PACK 7.1 101

GLOSARIO

ANS: Acrónimo para hacer referencia a un Sistema Neuronal Artificial (Artificial Neural Sistem).

BIAS: También llamado umbral, se considera como un peso adicional a la neurona al cual se le asocia una entrada permanente de valor 1 y se simboliza con la letra b.

BIOPSIA: Extirpación parcial del tejido para análisis y diagnóstico.

BIT: Unidad básica de información con la que trabajan los ordenadores. Puede tener dos estados (1,0).

BP: Backpropagation.

CLB: Bloque Lógico Configurable (Configurable Logic Block).

DESCRIPCIÓN COMPORTAMENTAL: Descripción de un sistema sin tener en cuenta el funcionamiento interno de este.

FPGA: Arreglo de Compuertas Programable por Campo (Field Programmable Gate Array).

FT: Función de transferencia o activación.

IOB: Input Output Block.

LUT: Look Up Table.

MAC: Unidades de multiplicación y acumulación.

MÁQUINAS VON NEUMANN: Son las clásicas máquinas de procesamiento secuencial.

MATLAB: Acrónimo de Matrix Laboratory. Lenguaje de programación para ingeniería desarrollado por MathWorks.

MLP: Perceptrón Multi Capa (Multi Layer Perceptron).

MODELSIM: Paquete computacional de la compañía Mentor Graphics que permite simular circuitos digitales descritos con HDLs.

NIBBLE: Grupo de 4 bits.

NPRTOOL: Comando de Matlab para acceder a bases de datos que permiten entrenar redes neuronales aplicables a reconocimiento de patrones.

PARALELISMO: El paralelismo de una red neuronal está directamente relacionado con el número de unidades de procesamiento que tiene la red.

PE: Procesador Elemental.

PESOS: Es un factor que se asocia a cada entrada de una neurona con el fin de excitarla o inhibirla y se simboliza con la letra W.

PIPELINING: Técnica utilizada en microprocesadores avanzados en la cual se comienza ejecutando una segunda instrucción antes que la primera haya sido finalizada.

PUERTO PARALELO. Punto de conexión de un ordenador que permite el traspaso de un grupo de bits a la vez.

PUERTO SERIAL: Punto de unión en el computador que hace posible el intercambio de bits (uno tras otro), con un equipo externo.

RED NEURONAL ARTIFICIAL: Son sistemas hardware o software de procesamiento que copian esquemáticamente la estructura neuronal del cerebro para tratar de reproducir sus capacidades y su acrónimo es RNA.

t: Valores objetivo de la red neuronal.

TAXONOMÍA DE FLYNN: Clasifica los sistemas digitales en 4 grupos dependiendo del flujo de datos y el flujo de instrucciones.

TOP DOWN: Metodología de diseño que parte de lo más general a lo más específico.

UART: Universal Asynchronous Receiver Transmitter. Es un componente del ordenador que maneja comunicaciones seriales asíncronas.

USB: Canal para el traspaso de datos universal.

VHDL: Lenguaje de descripción de Hardware de Alta Velocidad.

VLSI: Integración en escala muy grande.

INTRODUCCIÓN

El uso de redes neuronales es muy apropiado para abordar tareas en las cuales la información se presenta masiva, imprecisa y distorsionada, hecho por el cual esta técnica viene siendo cada día más utilizada. Por lo general su implementación es realizada mediante simulación software. Esto implica el uso de un sistema basado en microprocesador; caracterizado por utilizar un esquema de computo puramente secuencial que impide que sea explotada la capacidad de cálculo masivamente paralelo inherente a este tipo de redes, lo anterior resulta claro si se tiene en cuenta que el poder de procesamiento de una red neuronal no radica en la utilización de un potente procesador capaz de ejecutar una serie de instrucciones (una tras otra) a alta velocidad y con un alto grado de precisión, sino en la utilización de un grupo de procesadores muy sencillos y poco precisos (llamados neuronas) trabajando al mismo tiempo, esta es la razón por la cual la realización hardware resulta ser la mas indicada para la implementación de redes neuronales.

La tecnología de implementación más habitual para la realización hardware de redes neuronales es la microelectrónica VLSI, en forma de ASIC. Ésta tecnología, se adapta muy bien a la implementación de sistemas paralelos, como es el caso de las redes neuronales. El diseño de ASIC es una alternativa computacionalmente muy potente, pero así mismo su realización requiere una considerable inversión tanto en dinero como en tiempo de diseño, con lo cual se hace injustificable para el desarrollo rápido y económico de prototipos o series pequeñas.

Actualmente, gracias al surgimiento de tecnologías como los FPGAs, con la capacidad de reconfigurar su estructura hardware, se abre una nueva posibilidad de realizar implementaciones hardware de redes neuronales. Los FPGA permiten la creación de sistemas complejos en un corto tiempo de diseño , y por sus características resultan atractivos para la creación de sistemas neuronales electrónicos, no sólo por el hecho de que están en la capacidad de brindar un alto grado de paralelismo sino también por la flexibilidad que poseen, estos son fácilmente programables por el usuario mediante herramientas CAD, que permiten captura esquemática, síntesis y simulación a partir de un HDL (lenguaje de descripción de hardware) y aunque en comparación con los ASICs, su velocidad y densidad de puertas es menor, estos brindan la posibilidad de poderse reconfigurar. En base a lo anterior es como surge la necesidad de realizar un esfuerzo investigativo que apunte a aprovechar eficientemente los recursos y características de estas tecnologías a la hora de implementar redes neuronales, y por ende que contribuya al mejoramiento de los sistemas electrónicos actuales.

1. MARCO CONCEPTUAL

A lo largo de esta sección se exponen los conceptos fundamentales de la teoría de redes neuronales, especialmente los relacionados con la estructura de la neurona artificial, la arquitectura de red y modos de operación, enfocados hacia la solución de problemas prácticos. Posteriormente se describen las características principales que posee la Lógica Reconfigurable, profundizando en los dispositivos FPGA, lo cual permite que el lector comprenda las ventajas y desventajas que conlleva la utilización de estas tecnologías en la implementación de redes neuronales. Por último se termina con una breve perspectiva de la evolución de los FPGAs para generar una idea clara de la enorme potencia de procesamiento que tendrán en el futuro *las implementaciones de redes neuronales en hardware reconfigurable*.

1.1 REDES NEURONALES ARTIFICIALES (RNA)[1]

Las redes neuronales artificiales son sistemas de procesamiento inspirados en la estructura neuronal del cerebro para tratar de reproducir sus capacidades, los Sistemas Neuronales son capaces así de aprender de la experiencia a partir de las señales o datos provenientes del exterior. Su funcionamiento obedece a las siguientes reglas.

1. El procesamiento de información se lleva a cabo en muchos elementos simples con múltiples entradas y una única salida llamados neuronas (paralelismo de cálculo).

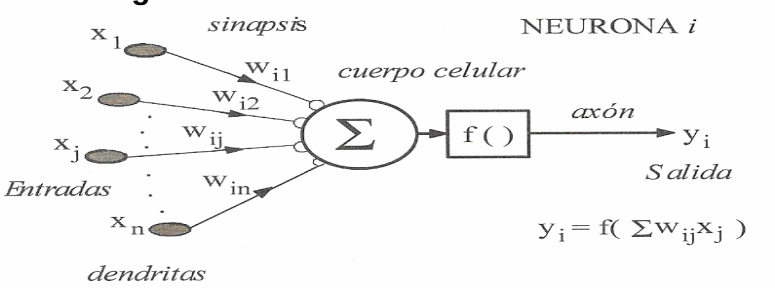
2. Las Neuronas envían señales entre ellas a través de un gran número de conexiones.

3. Cada conexión de una neurona *i* tiene asociado un factor que se multiplica con las señales de entrada ($x_1, x_2, \ldots, x_n$), estos factores se denominan pesos ($w_{i1}, w_{i2}, \ldots, w_{in}$). Las señales resultantes pasan posteriormente al cuerpo celular (ver figura 1).

4. Cada neurona *i* obtiene su salida y_i aplicando una función de activación o transferencia (FT), a la suma de las señales que entran al cuerpo celular. Las principales funciones de activación son mostradas en la tabla 1.

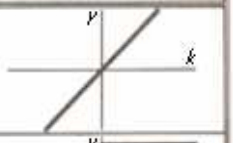
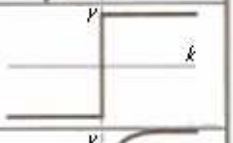
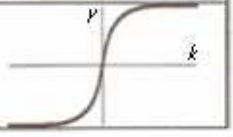
[1] SAMARASINGHE, Sandhya. Neural Networks for applied Sciences and engineering: from fundamental to complex pattern recognition. New York: Auerbach, 2006. p. 19.

Figura 1. Modelo de neurona estándar.



Fuente. Redes neuronales y sistemas borrosos.

Tabla 1. Funciones de activación habituales.

	Función	Rango	Gráfica
Identidad o Purelin	$y = k$	$(-\infty,+\infty)$	
Escalon o Hardlims	$y = Hardlims(k)$	$[-1,+1]$	
Sigmoidea o Tansig	$y = Tansig(k)$	$(1,+1)$	

Fuente. Redes neuronales y sistemas borrosos.

1.1.1 Estructura de un sistema neuronal artificial (ANS)[2]. El elemento más importante es la neurona (figura 1), el cual al agruparse con otras forman capas y a su vez la unión de varias capas constituye una red que junto con las interfaces de entrada y salida, más los módulos convencionales adicionales constituyen un sistema global. Un sistema neuronal está compuesto por los siguientes elementos (figura 2).

- Un Conjunto de Procesadores Elementales o Neuronas Artificiales.
- Un Patrón de Conectividad o Arquitectura. Puede ser de dos tipos, realimentado o unidireccional.

[2] DEL BRÍO, Bonifacio Martín y SANZ MOLINA, Alfredo. Redes Neuronales y Sistemas Borrosos. 3ra ed. México: Alfaomega, 2007. p. 13.

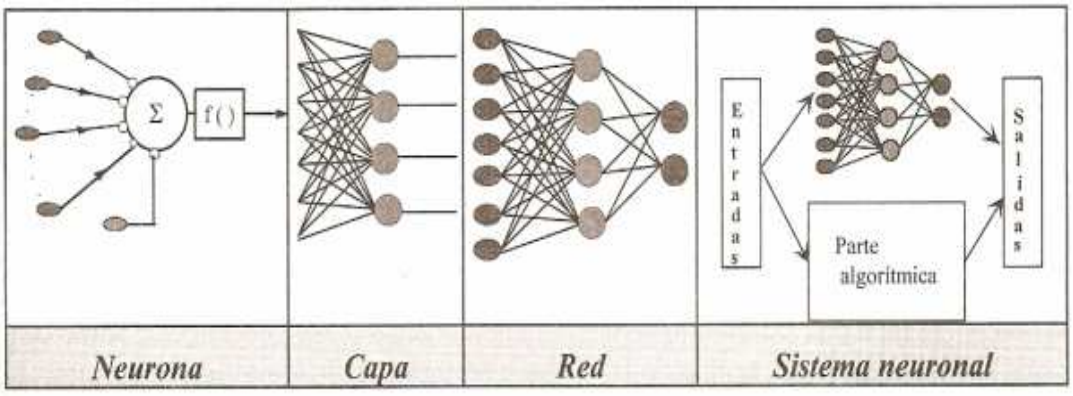
- Una Regla o Dinámica de Aprendizaje. También llamado algoritmo de aprendizaje. Permite determinar los pesos de cada conexión durante el entrenamiento.

- El Entorno Donde Opera.

1.1.2 Modos de operación de una red neuronal[3]. Las RNA pueden trabajar en dos modos: modo de aprendizaje o entrenamiento y modo de recuerdo o ejecución.

1.1.2.1 Modo de entrenamiento. Consiste en modificar los pesos sinápticos siguiendo una cierta regla de aprendizaje construida normalmente a partir del la optimización de una función de error, que mide la eficacia actual de la operación de la red.

Figura 2. Estructura jerárquica de un sistema basado en RNA.



Fuente. Redes neuronales y sistemas borrosos.

El anterior es un proceso iterativo que se repite una y otra vez hasta que la red alcanza el rendimiento deseado.

A continuación se presenta los tipos de entrenamiento más aplicados en redes neuronales.

- **Entrenamiento supervisado.**

La mayor parte de arquitecturas de RNA son entrenadas mediante métodos supervisados. En este tipo de entrenamiento, a la red neuronal se le presenta un

[3] Ibíd., p 30.

conjunto de entradas junto con sus respectivas salidas deseadas u objetivos, e iterativamente ésta ajusta sus pesos hasta que su salida tiende a ser la deseada, utilizando para ello información detallada del error que se comete en cada paso. Una vez que el entrenamiento termina, los pesos se fijan en sus valores obtenidos.

- **Entrenamiento no supervisado.**

A diferencia del aprendizaje supervisado, en el no supervisado no existe ningún maestro externo que indique si la red neuronal está operando correcta o incorrectamente. Así durante el entrenamiento, la red debe descubrir por sí misma, rasgos comunes, regularidades, correlaciones o categorías en los datos de entrada e incorporarlos en su estructura interna de conexiones (pesos). A este proceso se le llama auto-organización.

- **Aprendizaje hibrido.**

En este tipo de aprendizaje se hace uso de los dos tipos de aprendizaje anteriores los cuales tienen lugar normalmente en diferentes capas de la red.

- **Aprendizaje reforzado.**

Es un tipo intermedio entre el aprendizaje supervisado y no supervisado, consiste en indicarle a la red únicamente si está actuando en forma correcta o incorrecta, sin brindar ningún detalle más. En ocasiones se denomina aprendizaje por premio o castigo.

1.1.2.2 Modo de recuerdo. También llamado modo recuerdo (recall) o de ejecución Generalmente (aunque no en todos los modelos), una vez que el sistema ha sido entrenado, el aprendizaje se desconecta, por lo que los pesos y la estructura quedan fijos. Entonces se aplican nuevos datos de entrada con solución desconocida y se espera que la red esté preparada para dar una solución verdadera en base a lo aprendido durante el entrenamiento[4].

1.1.3 Capacidad de generalización[5]. Uno de los aspectos fundamentales de las RNA es su capacidad para generalizar. Por generalización se entiende la capacidad de la red de dar una respuesta correcta ante patrones que no han sido empleados en su entrenamiento. Una red neuronal bien entrenada generaliza, lo que significa que ha comprendido correctamente la mecánica del problema y no

[4] [en línea] Disponible en Internet: http://eltamiz.com/elcedazo/2008/10/21/ inteligencia-artificial-redes-neuronales/

[5] Ibíd., p. 75.

sólo los ejemplos concretos presentados. Hay que tener en cuenta que si se entrena una red hasta alcanzar un muy pequeño error de aprendizaje, la generalización se degrada. La explicación a esto es la siguiente; al principio la red se adapta progresivamente al conjunto de aprendizaje, acomodándose al problema y mejorando la generalización. Sin embargo, en un momento dado el sistema se ajusta demasiado a las particularidades de los patrones empleados en el entrenamiento, aprendiendo incluso el ruido entre ellos, cuando esto sucede se dice que hubo sobreaprendizaje o sobreajuste. Estos son dos aspectos que deben ser controlados adecuadamente para obtener un alto desempeño de red.

1.1.4 Patrones de entrenamiento[6]. Supóngase una red de *n* entradas y *S* salidas, entonces un patrón de entrenamiento está conformado por el conjunto de entradas ($x_1,x_2,\ldots,x_n$) y su respectivo conjunto de objetivos ($t_1,t_2,\ldots t_S$) (salidas deseadas de la red). Durante el entrenamiento, el aprendizaje se lleva a cabo mediante la presentación sucesiva de un set de patrones de entrenamiento. Cuando todo este paquete de datos ingresa a la red, se dice que ha transcurrido una *época*. Así, el proceso de aprendizaje se repite época tras época actualizando los pesos sinápticos hasta que se estabilicen en sus valores adecuados. Esta actualización es realizada generalmente, de dos formas:

1.1.4.1 Modo incremental. Los pesos sinápticos son actualizados a medida que cada patrón de entrenamiento sea presentado a la red.

1.1.4.2 Modo batch. Los pesos sinápticos se actualizan una vez haya entrado todo el conjunto completo de patrones de entrenamiento.

1.1.5 Clasificación de las redes neuronales artificiales[7]. Las redes neuronales artificiales se clasifican de acuerdo a dos de sus características más notables: *tipo de aprendizaje*, y *arquitectura de la red* (figura 3). Para efectos de este documento, se describen sólo los tres modelos más conocidos y habituales en las aplicaciones prácticas, como lo son: el Perceptrón, la Adalina y el Perceptrón Multi-Capa, profundizando en este último, ya que es el utilizado para lograr el objetivo final del presente proyecto.

[6] HOWARD DEMUTH, Mark Beale y HAGAN, Martin. Matlab Neural Network Toolbox™ 6 User's Guide, The Math Works Inc. 2009 [en línea] Disponible en Internet: (http://www.mathworks.com).

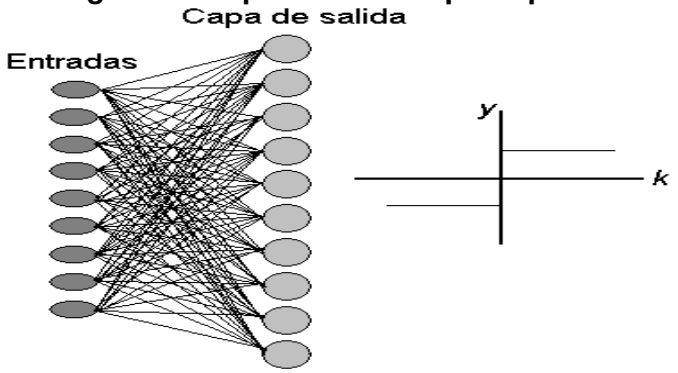
[7] DEL BRÍO y SANZ MOLINA. Op. cit., p. 32.

Figura 3. Clasificación de las redes neuronales artificiales.

Fuente. Redes neuronales y sistemas borrosos.

1.1.5.1 Perceptrón[8]. Este modelo tiene gran importancia histórica ya que fue el primero en poseer un mecanismo de entrenamiento que permite determinar automáticamente los pesos sinápticos que clasifican correctamente a un conjunto de patrones (linealmente separables) a partir de un conjunto de ejemplos. Estas redes consisten de una única capa de *S* neuronas conectadas a *n* entradas, como muestra la figura 4.

Figura 4. Arquitectura del perceptrón.



Un Perceptrón emplea principalmente dos funciones de transferencia (FT), *Hardlim* con salidas 1, 0 o *Hardlims* con salidas 1, -1; su uso depende del valor de salida que se espera, dando de esta manera, sólo salidas binarias. Una neurona con FT *Hardlim*, responde con 1 si el vector de entrada pertenece a la clase a la que representa y responde con 0 en caso contrario.

[8] BERTONA, Luís Federico. Entrenamiento De Redes Neuronales Basado En Algoritmos Evolutivos. Tesis de grado. 2005. p. 12. (Ingeniero de sistemas) Universidad de Buenos Aires.

La salida de la red de la figura 4 está determinada por:

$$y_i = Hardlims(\sum_{j=1}^{n} w_{i,j} x_j)$$

En ella, los sub-índices *i,j* indican que $w_{i,j}$ es el peso de conexión entre la *j*-ésima entrada y la *i*-ésima neurona[9].

El algoritmo de entrenamiento del perceptrón se encuentra dentro de los denominados algoritmos por corrección de errores. Este tipo de algoritmos ajustan los pesos de manera proporcional a la diferencia entre la salida actual proporcionada por la red y la salida objetivo, con el fin de minimizar el error producido por la red[10.]

1.1.5.2 Adalina / Madalina[11]. Su nombre proviene de Adaptive Linear Network (Red Adaptativa Lineal). La topología de la red Adalina (figura 5), es similar a la del Perceptrón, solo que en este caso la FT es de tipo lineal (purelin).

La red Adalina presenta la misma limitación del Perceptrón, en cuanto al tipo de problemas que pueden resolver. Ambas redes pueden resolver únicamente problemas linealmente separables, pero ésta utiliza como método de aprendizaje, la regla de Widrow-Hoff, también conocida como regla LMS (Least Mean Squares, mínimos cuadrados), el cual es más potente que el utilizado en el Perceptrón. Este algoritmo de entrenamiento realiza una actualización continua de los pesos sinápticos de acuerdo con la contribución de cada neurona sobre el error total de la red.

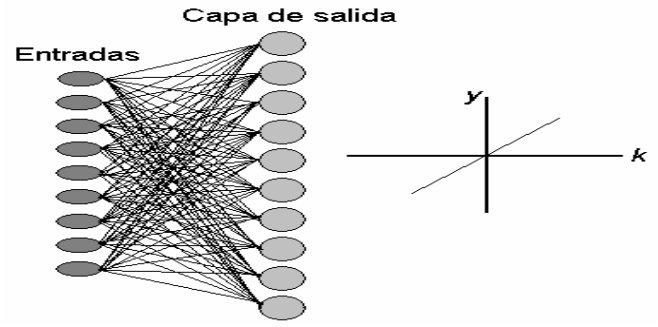
Existe una versión multicapa de la Adalina, denominada Madalina (Multiple Adalina), que consiste en una red neuronal con neuronas similares a las de la Adalina pero que contiene capas de neuronas ocultas.

[9] HOWARD DEMUTH, Mark Beale y HAGAN, Martin. Matlab Neural Network Toolbox™ 6 User's Guide, The Math Works Inc. 2009 [en línea] Disponible en Internet: (http://www.mathworks.com). P 108.

[10] WIDROW Bernard y LEHR, Michael A. Neural networks: Perceptrón, Madaline, and Backpropagation. 2003. p. 68.

[11] BERTONA, Op. cit. p.13.

Figura 5. Arquitectura de una red adalina.



La salida de la red está dada por:

$$y_i = Purelin(\sum_{j=1}^{n} w_{ij}x_j)$$

Al igual que el Perceptrón, la red Adalina es una red de aprendizaje supervisado que necesita conocer de antemano los valores asociados a cada entrada[12].

1.1.5.3 Perceptrón multicapa (MLP)[13]. Las Redes Perceptrón de una capa, mencionadas anteriormente, tienen una serie de limitaciones que les impiden realizar una gran variedad de aplicaciones, limitaciones que se pudieron solucionar con la inclusión de capas ocultas sobre estas redes, obteniendo de esta forma una red neuronal que se denomina Perceptrón Multicapa (MLP), que es quizá en la actualidad la red neuronal más utilizada.

Un Perceptrón multicapa es una red compuesta de varias capas de neuronas entre la entrada y la salida, dichas capas se unen de forma total hacia delante (feedforward), esto es, las entradas se unen con la primera capa y las salidas de ésta con la siguiente capa y así sucesivamente hasta la última capa (capa de salida).

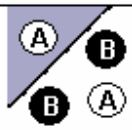
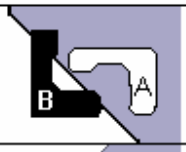
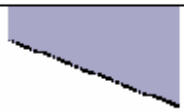
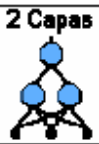
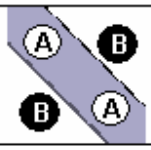
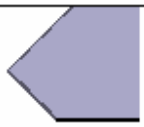
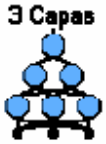
Esta red permite establecer regiones de decisión mucho más complejas (figura 6), así por ejemplo para una arquitectura de Perceptrón de una capa, la región de decisión es una recta, mientras que el Perceptrón multicapa con una única capa de neuronas ocultas puede discriminar regiones convexas. Por otra parte, el

[12] ACOSTA BUITRAGO, María Isabel y ZULUAGA MUÑOS, Camilo Alfonso. Tutorial sobre Redes Neuronales Aplicadas a Ingeniería Eléctrica y su Implementación en un Sitio Web. Colombia: Universidad Tecnológica de Pereira, 2000. p. 66.

[13] Ibíd., p. 63.

Perceptrón multicapa con dos capas de neuronas ocultas es capaz de discriminar regiones de forma arbitraria.

Figura 6. Regiones de decisión del MLP.

Estructura	Regiones de Decisión	Problema de la XOR	Clases con Regiones Mezcladas	Formas de Regiones más Generales
1 Capa 	Medio Plano Limitado por un Hiperplano			
2 Capas 	Regiones Cerradas o Convexas			
3 Capas 	Complejidad Arbitraria Limitada por el Número de Neuronas			

Fuente. Redes neuronales aplicadas en la ingeniería.

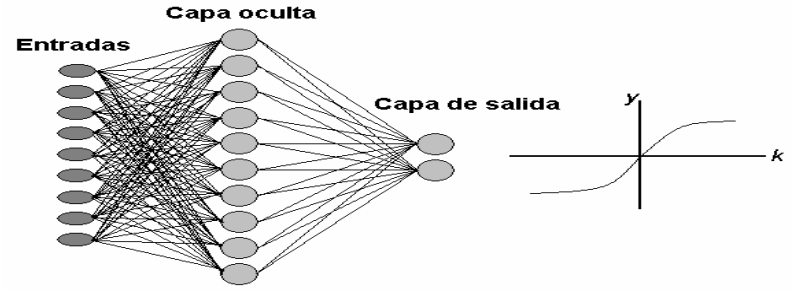
Para un correcto funcionamiento del MLP no es suficiente con realizar algunas modificaciones a la topología de la red, sino que se requiere de la elección de un buen método de aprendizaje como de la función de transferencia correcta; por lo general, se elije el algoritmo de aprendizaje supervisado Backpropagation (BP) y una función de transferencia diferenciable y no lineal como lo es la sigmoidea (tansig). La FT sigmoidea es utilizada en la mayoría de las redes MLP en sus capas ocultas, ya que le permite a los ANS aprender las variaciones no lineales de los distintos tipos de ambientes[14].

La figura 7, muestra la arquitectura abreviada de un MLP. Tanto las neuronas de la capa oculta como las neuronas de la capa de salida usan como regla de propagación la suma ponderada de sus entradas con los pesos sinápticos y sobre esa suma ponderada se aplica una función de transferencia de tipo tansig, que es acotada en respuesta[15].

[14] HOWARD DEMUTH y HAGAN. Op. cit., p. 170.

[15] DEL BRÍO y SANZ MOLINA. Op. cit., p. 63.

Figura 7. Arquitectura de un MLP de dos capas.



El modelo matemático de este tipo de redes, tiene gran importancia en el presente trabajo y se analiza más profundamente en la sección 2.1.

1.1.6 Algoritmo backpropagation[16]. Es el algoritmo de entrenamiento más utilizado en redes MLP. Es un método de aprendizaje supervisado de gradiente descendente, en el que se distinguen claramente dos fases. Primero se calcula la salida de la red para un patrón de entrada dado; esta salida se compara con la salida deseada, u objetivo, a partir de la cual se obtiene un error que determina la eficacia actual de la operación de la red. En la segunda fase se modifican los pesos en función de minimizar dicho error, mediante un procedimiento de propagación hacia atrás, este proceso se repite hasta cumplir con un criterio de parada (…)[17].

Expuestos los fundamentos básicos de las redes neuronales, a continuación se procede a describir las herramientas relacionadas con su puesta en práctica en esta investigación, las cuales quedan enmarcadas dentro de la llamada lógica reconfigurable.

1.2 LÓGICA RECONFIGURABLE[18]

La lógica reconfigurable, es un escenario del diseño digital que coloca a disposición de los diseñadores herramientas de tipo software y hardware para la realización de aplicaciones mucho más robustas en menor tiempo y a menores costos. Involucra un paradigma de diseño basado en metodologías software, que

[16] BERTONA. Op. cit., p 19.

[17] FREEMAN James y SKAPURA, David M. Neural Networks Algorithms, Applications, and Programming Techniques. Wokingham: Addison-Wesley, 1991.

[18] HERNANDEZ PEREZ, Alberto. Introducción al Lenguaje de Descripción de Hardware VHDL. Cuba: Centro de Investigaciones en Microelectrónica, CUJAE, 2004. p. 12.

conduce a una categoría de abstracción superior (Hardware), y que permite simplificar los procesos de verificación mediante la simulación.

Entre los dispositivos hardware y las herramientas software que hacen parte de la lógica reconfigurable se puede mencionar:

- Lenguajes de descripción de hardware (HDL).
- Codiseño hardware/software.
- Dispositivos programables.

1.2.1 Lenguajes de descripción de hardware (HDLs). Un lenguaje de descripción de hardware (HDL) es una herramienta que sirve para describir el comportamiento y la arquitectura de un circuito o sistema electrónico. Al utilizar un HDL para modelar un sistema, es importante recordar que se está modelando hardware, y no escribiendo software.

Los HDLs permiten describir la concurrencia y paralelismo que posee un sistema hardware (muchas cosas y muchas señales cambian al mismo tiempo). En dichos sistemas el tiempo es una variable importante; un cambio en el tiempo de propagación de una compuerta, el retardo de una traza en una plaqueta, o el no cumplir con los tiempos de establecimiento de un circuito, puede cambiar de manera radical el comportamiento del sistema digital que se esté realizando.

Usando HDLs, los sistemas digitales pueden describirse a diferentes niveles de abstracción. Estos modelos pueden ir desde el nivel abstracto del comportamiento general de un circuito hasta el nivel de interconexión de compuertas lógicas.

Uno de los aspectos interesantes de los HDLs es que permiten utilizar módulos ya creados, en nuevos diseños, lo que permite, por ejemplo, realizar conexiones en cascada, facilitando y optimizando el diseño[19].

Aunque hay muchos lenguajes de descripción de hardware, dos predominan actualmente en el mundo del desarrollo digital: Verilog y VHDL. Existen otros, desarrollados por los distintos proveedores de herramientas o PLD, pero en la actualidad casi todos los fabricantes de PLD y proveedores de herramientas de desarrollo proveen soporte para VHDL y Verilog[20].

[19] ARMSTRONG, J R y GRAY, F G. Structure Logic Desing with VHDL. London England: Prentice Hall, 1993. p. 103.

[20] ASHENDEN, Peter J. The Designer's Guide to VHDL. 2da ed. Morgan Kaufman, 2002. p. 256.

1.2.2 Codiseño hardware / software[21]. El término de codiseño Hardware / software es relativamente reciente, de principios de la década de los 90. Este término describe la confluencia de dos problemas en el diseño de circuitos integrados. Por una parte, está la arquitectura hardware y todas las tecnologías disponibles para su implementación y por otra, el diseño software que involucra los sistemas operativos, los lenguajes de programación, compiladores, herramientas de modelado, simulación y evaluación.

Con esta técnica es posible dividir las tareas a realizar (particionado hardware/software) de manera que parte de ellas son implementadas mediante un hardware de propósito específico y otras mediante la programación de un microprocesador de propósito general[22]. De tal forma que las tareas más críticas se implementan en hardware mientras que aquellas que se adaptan mejor a la ejecución de un microprocesador y en las que se requiere mayor flexibilidad se programan en software.

1.2.3 Dispositivos lógicos programables (PLDs). Son dispositivos de propósito general que contienen circuitos lógicos que se pueden programar. En la actualidad representan a uno de los sectores que más rápido crecimiento experimenta en la industria de los semiconductores debido a que permiten la creación de sistemas mucho más complejos en comparación con los sistemas creados mediante la actual lógica discreta.

La fabricación de dispositivos de lógica programable se fundamenta en el hecho de que cualquier función lógica se puede crear mediante una suma de productos (funcionalidad completa) y en el empleo de bloques lógicos que almacenan los datos de salida del circuito en lugar de implementar físicamente su ecuación booleana (celdas de funciones universales)[23].

Las características principales de los PLDs son:

- Dispositivos con diversos recursos lógicos incorporados.
- Configurables por el usuario.

---

[21] LAPLANTE, P. Real-time systems design and analysis. An engineer's handbook. IEEE Press: 1992. p. 198.

[22] MAGDALENO, E.; RODRÍGUEZ, M.; AYALA, A.; MENDOZA, B. y RODRÍGUEZ, S. Metodología para el Aprendizaje de Sistemas Electrónicos Digitales y su Diseño. Bogotá: TAEE, 2004. p. 205.

[23] [en línea] Disponible en Internet: http:/gemini.udistrital.edu.co/ comunidad/ profesores/jruiz/jairocd

- Poseen Interconexiones programables.
- Requieren herramientas CAD.

Los dispositivos lógicos programables (PLDs) forman parte de los circuitos de alto nivel de integración, que incluyen, entre otros, los PLDs simples (SPLDs), los PLDs complejos (CPLDs) y los arreglos de compuertas programables por campo (FPGAs). Por su importancia en esta investigación los últimos se abordan en forma más profunda a continuación.

1.3 FPGA[24] (FIELD PROGRAMMABLE GATE ARRAY)

Un FPGA es un arreglo de bloques lógicos, colocados en una infraestructura de interconexiones programables, fueron introducidos por Xilinx en 1985 y hoy en día son el dispositivo programable por usuario de más amplio espectro de aplicación. Es posible programar la funcionalidad de sus bloques lógicos, las interconexiones entre bloques y las conexiones entre entradas y salidas. Representan uno de los últimos avances en tecnología PLDs; algo muy importante de estos dispositivos es que se reconfiguran con un programa, a diferencia de lo que normalmente se conoce como sistema programado (microcontrolador, microprocesador etc), en donde un hardware fijo es capaz de interpretar y ejecutar un algoritmo, en los FPGA lo que se tiene es un hardware que se configura realizando conexiones físicas que son especificadas por un programa o cadena de configuración.

Estos dispositivos poseen características que los hacen atractivos para la implementación de redes neuronales, puesto que son capaces de brindar el paralelismo requerido, vienen dotados de altas capacidades en memoria y con bloques aritméticos dedicados; además su reconfigurabilidad puede ser aprovechada para optimizar operaciones de multiplicación por medio de tablas de búsqueda internas LUTs (look-up tables). No obstante debe aclararse que las redes neuronales debido a las númerosas conexiones y operaciones que realizan internamente consumen un elevado número de recursos hardware en el FPGA. Hace algunos años esto fue un limitante importante que causaba que los FPGAs sean inaplicables en la solución de problemas complejos mediante redes neuronales. Dicha limitación está siendo superada gracias a los permanentes avances en tecnología microelectrónica que permiten integrar millones de puertas en un pequeño chip.

[24] GÜICHAL, Guillermo. Diseño Digital Usando Lógica Programable. México: Universidad Tecnológica Nacional Facultad Regional Bahía Blanca, 2005. p. 47.

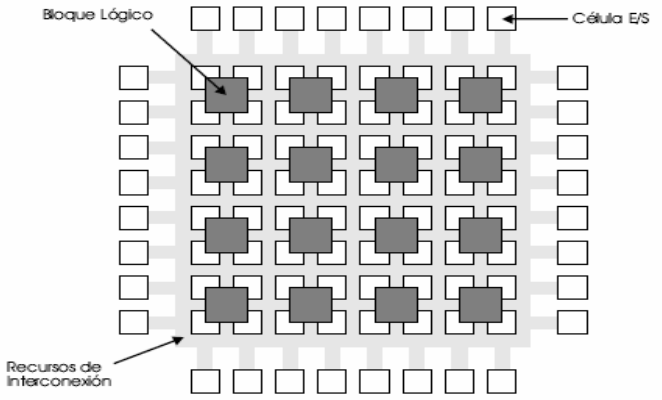
Los elementos básicos constituyentes de un FPGA se pueden ver en la figura 8 y son los siguientes:

- **Bloques lógicos configurables (CLBs):** Estos son los bloques básicos que se utilizan en la implementación de un circuito digital. Cada CLB contiene circuitos que le permiten realizar operaciones aritméticas eficientes y en forma concurrente (lo que convierte a los FPGAs en herramientas verdaderamente óptimas para la implementación de redes neuronales).

- **Bloques de entrada - salida (IOBs):** Estos bloques proveen la interface entre los "pines" del integrado y la lógica interna.

- **Recursos de interconexión:** Es una estructura versátil y multi-nivel de interconexión entre los otros componentes de la FPGA.

Figura 8. Estructura básica de los FPGA.



Fuente. Diseño digital usando lógica programable.

El proceso de programación no es único, sino que se puede realizar mediante diferentes tecnologías, como son células RAM estáticas, transistores, memorias EPROM y EEPROM, etc. En el caso de las FPGAs de XILINX los elementos de programación se basan en células de memoria RAM que controlan transistores de paso, puertas de transmisión o multiplexores.
Los FPGAs permiten realizar una gran cantidad de aplicaciones y en muy diversas áreas, sobre todo en aquellas que requieren un alto grado de paralelismo. Entre éstas se tienen el procesamiento digital de señales, radio definido por software, sistemas aeroespaciales y de defensa, prototipos de ASICs, sistemas de imágenes para medicina, sistemas de visión para computadoras, reconocimiento de voz, bio-informática, emulación de hardware en computadora, entre otras.

1.3.1 Programación de los FPGAs. La tarea del programador es definir la función lógica que realiza cada uno de los *CLBs*, seleccionar el modo de trabajo de cada *IOB* e interconectarlos.

El diseñador cuenta con la ayuda de *entornos de desarrollo* especializados en el diseño de sistemas a implementarse en FPGA. Un diseño puede ser capturado ya sea como esquemático, o haciendo uso de un lenguaje de descripción de hardware (HDL). Los HDLs más utilizados son VHDL y Verilog[25].

1.3.2 Evolución de los FPGAs. Pocos campos de la ciencia han avanzado de una forma tan vertiginosa como lo ha hecho la Electrónica, siendo sobre todo más evidente en lo que al diseño digital se refiere. Entre esos avances digitales se encuentra la tecnología FPGA.

El número de recursos internos de los FPGAs ha crecido notablemente en los últimos años, generándose dispositivos cada vez más preparados para distintas aplicaciones. Gracias a ello, es posible que estos dispositivos hayan pasado de ser usados inicialmente como elementos coprocesadores, a ser una poderosa plataforma tecnológica sobre la cual funcionan complejos sistemas de respuesta en tiempo real. Esto ha sido debido principalmente a las mejoras producidas en los procesos de fabricación de circuitos integrados y a la mejora del software de síntesis e implementación.

La mejora en los dispositivos reconfigurables ha posibilitado la implementación de sistemas digitales de alto nivel, comúnmente denominados como SOC (*System On Chip*), con un ahorro tanto económico como en tiempo de diseño. Económico porque el número de circuitos integrados se reduce al poder integrar en un FPGA númerosos sistemas digitales. Si el análisis se realiza desde un punto de vista de tiempo de diseño, los tiempos de elaboración de un proyecto también se reducen al integrar todo en un FPGA.

Uno de los principales problemas al que se enfrentan las personas que trabajan con FPGAs es cual escoger, ya que existen varios fabricantes que disponen de diversas ventajas. Los diseñadores coinciden en afirmar que existen tres fabricantes mayoritarios en la distribución de FPGAs y software de soporte: *Xilinx, Altera* y *Actel*. En el mercado mundial se pueden encontrar otros tantos con producciones menores pero que figuran también como FPGAs útiles: *Lucent, Texas Instruments, Philips, QuickLogic, Cypress, Atmel*, etc. El siguiente apartado se enfoca a los FPGAs de Xilinx, ya que el dispositivo utilizado en este proyecto pertenece a éste fabricante; además se da una breve introducción a las diferentes

[25] BASIL, M y SUARDÍAZ MURO, Juan. Nuevas Tendencias En El Diseño Electrónico Digital. Madrid España: Codiseño Hardware/Software, Universidad Alfonso X El Sabio, 2007. p. 98.

familias lógicas y a sus principales características, para así poder dar una idea clara de cuáles son los FPGAs más avanzados que tiene Xilinx en la actualidad[26].

1.3.2.1 FPGAs de Xilinx[27]. Xilinx está considerado como líder entre los fabricantes de FPGAs a nivel mundial. Sus FPGAs están basados en la tecnología SRAM. Sus principales familias son: *XC3000, XC4000, XC Virtex*, *y XC Spartan*. La serie *XC Virtex*, o tan solo *Virtex*, es la más nueva de todas las familias de Xilinx. Se dice que los dispositivos pertenecientes a esta familia son los más rápidos (velocidades de trabajo de hasta 250 Mhz), densos en compuertas y menor consumo de potencia, pero por lo mismo son los más costosos.
La serie *Spartan* surgió como una opción para sustituir diseños probados de menos de 15.000 compuertas por dispositivos de bajo costo y alto desempeño, aunque llegan hasta aproximadamente 5 millones de puertas. Las series estándar *XC3000* y su sucesora, la *4000*, muestran la mayor parte de las características funcionales de los FPGAs de Xilinx, pero con la gran desventaja que son dispositivos de baja densidad de puertas, por lo cual son utilizadas para realizar prototipos de bajo impacto. La tabla 2 ilustra una breve comparación entre las familias de FPGAs de Xilinx.

[26] Ibíd., p. 120.

[27] HERRERA LOZADA, Juan Carlos. México: Tecnologías Programables, 2007. p. 29.

Tabla 2. Familias de FPGAs de Xilinx.

Familia	Año de creación	Número máx. de puertas	Novedades
XC2000	1985	1800	Configuración SRAM
XC3000	1987	7500	Mayor número de puertas
XC4000/ Spartan	1991	180000	Memoria RAM
Virtex/ Spartan II	1998	3M2/600K	BRAMs, DLLs
Virtex II	2000	8 M	DCMs. Multiplicadores
Spartan 3	2003	5 M	Bajo Coste
Virtex II-Pro	2002	4 M	Power PC, Rocket I/O
Virtex 4	2005	8 M	Ethernet MACs, DSP Slice
Virtex 5	2006	11 M	DSP Slice Avanzada, BRAM 32KBits

Fuente. Arquitectura basada en FPGA, para la detección de objetos.

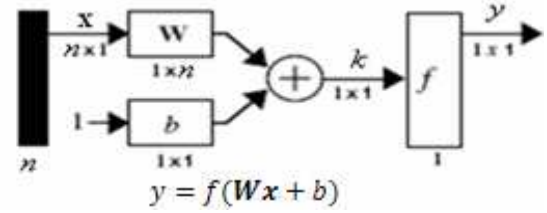
2. ETAPA INVESTIGATIVA

Para implementar un sistema neuronal en hardware es necesario primeramente adquirir algunos conceptos básicos que permitan definir sus principales características, por tanto en esta etapa del proyecto se discuten temas relacionados con el modelo matemático, tipos de paralelismo, manejo de la aritmética interna, modos de entrenamiento y bloques básicos de un sistema neuronal.

2.1 DEFINICIÓN DEL MODELO MATEMÁTICO DE UNA RED NEURONAL[28].

2.1.1 Representación matricial de una neurona estándar. Matricialmente, la salida *y* de una neurona estándar se puede modelar por medio de una función de activación *f*, que tiene como argumento *k*, la multiplicación de un vector de entrada **x** (vector columna compuesto por *n* elementos, $x_1, x_2, x_3 \ldots x_n$), por una matriz de pesos **W** (compuesta por los pesos $w_{11}, w_{12}, w_1 \ldots w_{1n}$), más un parámetro adicional denominado *Bias* o *Umbral* (puede considerarse como un peso adicional, al cual se le asocia permanentemente una entrada de valor 1), ver figura 9.

Figura 9. Modelo matricial de una neurona estándar.



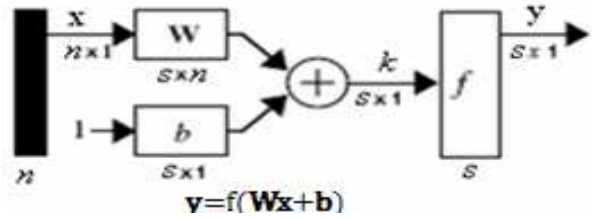
Fuente. Matlab neural network toolbox.

Nótese que en esta representación la matriz **W** tiene un único renglón, debido a que se está representando una única neurona.

2.1.2 Representación matricial de una capa de neuronas. En base a la representación anterior, de neurona estándar, una capa de *S* neuronas puede ser representada como se muestra en la figura 10.

[28] HOWARD DEMUTH, y HAGAN. Op. cit., p. 90.

Figura 10. Modelo matricial de una capa de neuronas.



Fuente. Matlab neural network toolbox.

Donde:

x, es un vector de entrada de *n* elementos.
W, es la matriz de pesos.
b, es el vector que contiene los Bias de cada neurona.
y, es la salida de *S* elementos.

Cabe destacar que la única diferencia entre representación de una neurona estándar y una capa de neuronas, es el tamaño de la matriz **W** y el vector **b**, ya que cada renglón de la matriz **W** representa el conjunto de pesos para cada neurona en la capa y cada elemento del vector **b** contiene el respectivo Bias de las neuronas.

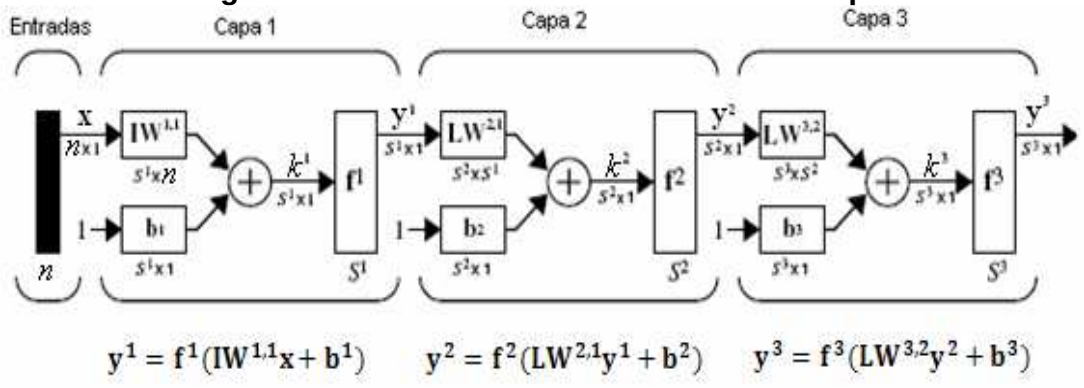
2.1.3 Representación matricial de redes multicapa. Para describir redes que poseen varias capas, se debe extender la notación hasta ahora utilizada, es necesario primero, hacer una distinción entre las matrices de pesos conectadas a las entradas y las matrices de pesos conectadas entre capas, y segundo, identificar el origen y el destino de las señales que entran a dichas capas.

Las matrices de pesos conectadas a la entrada de la red son llamadas *Pesos de Entrada* **IW** y las matrices de pesos conectadas a las salidas de otras capas son llamadas *pesos de capa* **LW**. Se utilizan superíndices para indicar el origen (segundo índice) y el destino (primer índice) de las señales que entran a las matrices de pesos.

En la figura 11, se muestra una red neuronal de tres capas, haciendo uso de la notación extendida, anteriormente mencionada.

Cada capa tiene asociado una matriz de pesos **W**, un vector columna **b**, y un vector columna de salida **y**. Para distinguir entre el vector de pesos, vectores de salida, etc, de cada una de las capas, se utiliza un superíndice en la variable de interés.

Figura 11. Modelo matricial de redes multicapa.



Fuente. Matlab neural network toolbox.

La red mostrada arriba tiene *n* entradas, S^1 neuronas en la primera capa, S^2 neuronas en la segunda capa y S^3 neuronas en la tercera capa. Nótese que las salidas de cada capa intermedia son las entradas a la siguiente capa, así la capa 2 puede ser analizada como una red de una capa con S^1 entradas, S^2 neuronas y una matriz de pesos $\mathbf{LW}^{2,1}$ de S^1xS^2 elementos. La entrada a la capa 2 es $\mathbf{y}^1$ y la salida es $\mathbf{y}^2$**.** Ahora que todos los elementos de la capa 2 han sido identificados ésta puede ser tratada como una red de una única capa de neuronas, en la figura 11, se puede observar las ecuaciones para cada una de las capas, (esta metodología puede ser aplicada a cualquier tipo de red).

Con lo mencionado hasta el momento, se puede definir matemáticamente la red neuronal de la figura 11, como sigue.

$$\mathbf{y}^3 = \mathbf{f}^3(\mathbf{LW}^{3,2}\mathbf{f}^2(\mathbf{LW}^{2,1}\mathbf{f}^1(\mathbf{IW}^{1,1}\mathbf{x} + \mathbf{b}^1) + \mathbf{b}^2)\mathbf{b}^3)$$

Las capas que producen la salida de una red son llamadas capas de salida, y las demás son llamas capas ocultas. Aunque aquí no sea considerado así, se debe resaltar que algunos autores consideran al vector de entrada como una capa mas, la cual es llamada capa de entrada.

Habiendo definido la representación matemática que modela a una red neuronal se procede a discutir algunos aspectos relacionados con su implementación. Estos son necesarios para hacer un uso adecuado de los recursos hardware que poseen las tecnologías reconfigurables.

2.2 PARALELISMO EN REDES NEURONALES[29]

Los altos requerimientos computacionales de las tareas para las cuales está indicado el uso de redes neuronales, hacen que las arquitecturas Von Neumann (en las cuales se basan los sistemas digitales convencionales) no sean muy apropiadas, por ello la introducción de sistemas de cálculo paralelo resulta necesaria si se quiere lograr respuestas en tiempo real. De lo anterior se desprende el concepto de paralelismo. El paralelismo en una red neuronal está directamente relacionado con el número de unidades de proceso presentes en la red (también denominadas, *procesadores elementales o PEs*), los cuales son elementos secuenciales de la arquitectura encargados de realizar las operaciones aritméticas.

Las redes neuronales pueden ser desarrolladas en un FPGA a diferentes niveles de paralelismo, analizar cada uno de estos, resulta de gran importancia a la hora de escoger tanto las estructuras hardware más adecuadas como los métodos que sacan un mayor provecho de los recursos del dispositivo. El paralelismo varía enormemente de nivel a nivel y una aplicación específica puede utilizar un grado de paralelismo intermedio, dependiendo de los costos (en términos de CLBs) y el rendimiento deseado, así por ejemplo si la velocidad de un PE es baja, este problema puede ser superado con la utilización de dos o más PEs trabajando en paralelo. Los niveles de paralelismo más destacados son los siguientes.

2.2.1 Paralelismo de capa. En redes de múltiples capas, éstas pueden ser procesadas en paralelo, para lo cual se utiliza un único PE por capa, el paralelismo a este nivel es por lo general bajo, y por consiguiente de limitado valor, pero puede sacar provecho de técnicas como *pipelining*.

2.2.2 Paralelismo de nodo. Éste es el nivel que corresponde a neuronas individuales, utiliza un PE para cada neurona, su implementación puede no ser posible para aplicaciones que requieran un elevado número de ellas.

2.2.3 Paralelismo de sinapsis. Es el paralelismo de más alto nivel alcanzado en redes neuronales. Tiene como característica utilizar para cada neurona, elementos de proceso dotados con capacidad de cómputo paralelo, cualidad que poseen por el hecho de utilizar un número de multiplicadores igual al número de entradas en la neurona. Las aplicaciones realizadas a este nivel son muy potentes, pero así mismo el gasto de recursos es enorme.

[29] AMOS, R y OMONDI, Jagath C. Rajapakse, FPGA Implementations of Neural Networks. Holanda: Springer, 2006. p. 12.

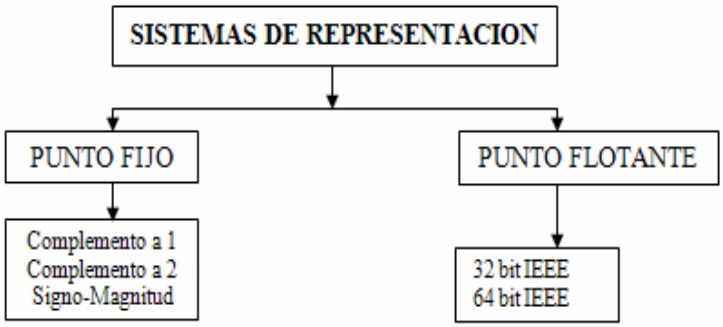
2.2.4 Paralelismo sistémico. Es un paralelismo de nivel medio y resulta conveniente cuando los recursos disponibles son pocos. Consiste en múltiples procesadores trabajando concurrentemente para determinar la respuesta general de la red; es decir, cada capa se computa con paralelismo a nivel de nodo, pero las diferentes capas del sistema son evaluadas en tiempos diferentes. Esto involucra la utilización de módulos de almacenamiento para guardar datos intermedios. Las máquinas *SIMD* son ejemplo de este tipo de paralelismo.

2.3 ARITMÉTICA EN REDES NEURONALES

Para realizar implementaciones hardware de redes neuronales, se deben considerar aspectos referentes a la aritmética interna del diseño, estos incluyen la representación de datos, la forma en cómo se llevan a cabo las multiplicaciones y sumas de productos y la implementación de las funciones de activación.

2.3.1 Representación de datos. La forma de representar los datos, es un aspecto en el cual se debe tener mucho cuidado, y preferiblemente debe ser definido en las primeras etapas de un proyecto. La decisión de escoger la representación en punto fijo o en punto flotante, como más conveniente en una determinada aplicación, en general se puede tomar asumiendo que las implementaciones en punto fijo alcanzan mas velocidad y gastan menos recursos hardware, pero las precisiones que manejan son menores, mientras que en punto flotante pueden alcanzar un rango dinámico más alto y por lo tanto mayor precisión (lo cual es atractivo si se quieren utilizar algoritmos complicados) pero gastan muchos más recursos en el dispositivo. Ambos sistemas han sido estandarizados, pero pueden ser modificados a conveniencia del diseñador, para una aplicación específica. La figura 12, muestra los sistemas de representación de datos mas utilizados (…)[30].

Figura 12. Sistemas de representación de datos.



[30] FLOYD Tomas L. Fundamentos de Sistemas Digitales. 7ma ed. Madrid: Prentice Hall, 2002.

2.3.1.1 Mínimo rango de precisión permitido[31]. Un diseño en FPGA debe realizarse en forma tal, que haga uso lo más eficientemente posible de los recursos hardware que posee el dispositivo. En términos de un óptimo balance entre rango de precisión y área utilizada, esto puede ser logrado determinando *la mínima precisión permitida* y *el mínimo rango permitido*, para los cuales el criterio es minimizar el área hardware utilizada, sin sacrificar calidad de rendimiento. Esos dos conceptos combinados son referidos como el *mínimo rango de precisión permitido*.

Determinar el mínimo rango de precisión permitido puede traducirse a hallar la máxima cantidad de incertidumbre (error de cuantización debido a limitada precisión), que una determinada aplicación puede resistir antes de que su rendimiento comience a degradarse.

Hay estudios que establecen 8 bits para salidas de funciones de activación y 16 bits para pesos[32]. Sabiendo esto, el aspecto crítico es determinar cuándo menor precisión debe ser utilizada debido a consideraciones de rendimiento y costo. En este caso un cuidadoso análisis debe ser realizado. Debe tenerse en cuenta que gracias a las ventajas de los FPGAs y del diseño VHDL éste es un parámetro que fácilmente puede ser modificado hasta encontrar la mejor implementación.

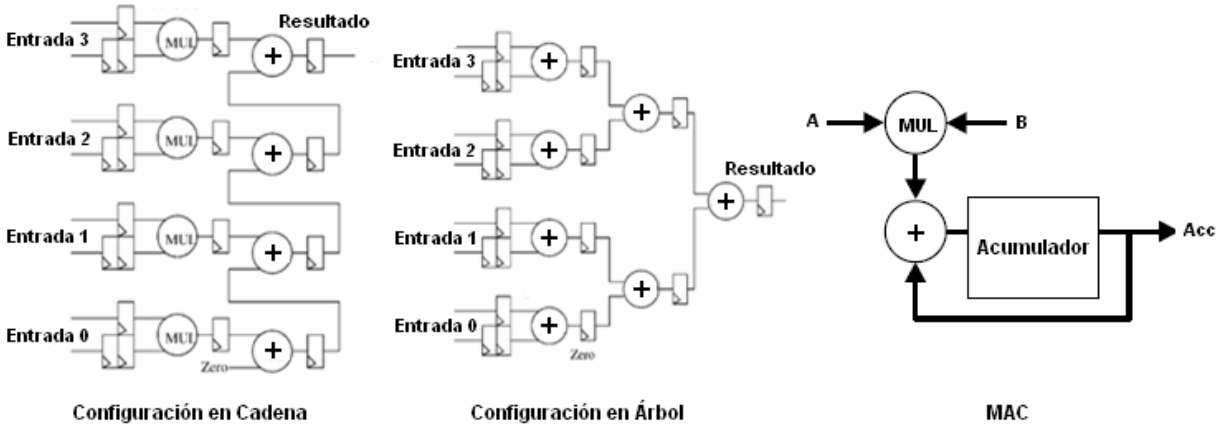
2.3.2 Procesamiento de suma de productos[33]. Todas las señales que entran a una neurona artificial son multiplicadas por sus respectivos pesos y posteriormente todos estos productos son sumados en el cuerpo de la neurona. Este proceso requiere el uso de bloques aritméticos que pueden ser dispuestos de diferentes maneras dentro del diseño, entre estas disposiciones se destacan la configuración en cadena, configuración en árbol y mediante unidades de multiplicación y acumulación (MAC), la escogencia de una u otra puede variar el grado de paralelismo deseado; esto se puede observar en la figura 13.

[31] AMOS, R y OMONDI. Op. cit., p. 40.

[32] HOLT, J. L. and HWANG, J. N. Finite-precision error analysis of neural network hardware implementations. Washington, DC, USA: IEEE Transactions on Computers. 1993. p 290.

[33] AMOS, R y OMONDI. Op. cit., p. 17.

Figura 13. Configuraciones habituales para suma de productos.



Fuente. FPGA implementations of neural networks.

2.3.3 Funciones de activación. Son usadas para determinar la salida de una neurona, tomando como argumento de entrada, la *suma de productos internos*, hay muchos tipos de estas funciones entre las cuales se destacan, sigmoidea, hard limit y purelin.

Muchas técnicas existen para evaluar estas funciones, entre las que se citan Algoritmos CORDIC, Aproximaciones Polinómicas, Expansión en Series de Taylor y Métodos por Manejo de Tablas. En términos de rendimiento precisión y costo, todos tienen ventajas y desventajas. Se quiere destacar entre estos el método por manejo de tablas, el cual optimiza la implementación de estas funciones mediante el uso de tablas internas LUTs, lo cual resulta muy conveniente teniendo en cuenta que los FPGAs vienen equipados con una gran capacidad de memoria.

2.3.3.1 Método por manejo de tablas[34]. Debido a que es necesario usar valores con limitada precisión se hace necesario discretizar tres grupos diferentes de variables, entradas y salidas de neuronas (ambas con el mismo número de bits), pesos sinápticos y el argumento de entrada de la función de transferencia. Cada uno de estos grupos tiene diferentes rangos especificados por el número de bits de la variable discreta y del máximo valor absoluto de la correspondiente variable continua.

En implementaciones digitales, únicamente la función hard limit puede ser exactamente evaluada, las demás como es el caso de las funciones purelin y sigmoidea tienen que ser aproximadas.

[34] Ibíd., p. 280.

A continuación se describe como obtener los datos que se guardaran en la LUT para la función de activación sigmoidea y la manera de usarlos para obtener la salida de la neurona.

Con el propósito de lograr generalidad, una función de activación sigmoidea *tansig(k)* es definida dependiendo de tres parámetros, f_0' (derivada en k=0), f_{max} (valor máximo de *tansig(k)*), y f_{min} (valor minimo de *tansig(k)*). Por simplicidad se define f_R como f_{max}-f_{min}.

$$Tansig(k) = \frac{f_R}{1+e^{-\frac{4f_0' k}{f_R}}} + f_{min}$$

$f_0'=1/2$, $f_{max}=1$, $f_{min}=-1$

Ahora se definen algunos parámetros adicionales:

- *n:* máximo número de entradas a una neurona (incluye el Bias).
- M_z: máximo valor absoluto de las entradas y salidas continúas de la red.
- p_z : número de bits de las salidas y entradas discretas de la red.
- M_w: máximo valor absoluto de los pesos continuos.
- p_w número de bits de los pesos discretos.
- M_s: máximo valor absoluto de la suma de productos continúa.

$$M_s = nM_zM_w$$

- N_s: número de bits de la suma de productos discreta.

$$N_s = \left[log_2\left(n(2^{p_z-1}-1)(2^{p_w-1}-1)\right)\right] + 1$$

- **Direccionamiento de la LUT.**

Para poder acceder a los datos que contiene la LUT, el procedimiento consiste en dividir en dos partes los N_s bits que representan la suma de productos. La primera parte se compone de los p_s bits más significativos (usados como entrada de la LUT) y la segunda de los m_s bits menos significativos.

$$N_s = p_s + m_s \qquad 1 \le p_s \le N_s$$

Entre más grande sea p_s, más exacta es la función de activación; se debe encontrar un valor para p_s (el cual se nombrara como valor optimo p_{Sop}), éste es el minimo p_s que satisface la condición de que todos los posibles valores de salida de la función de activación discreta están presentes en la LUT. Una posible aproximación para p_{Sop} es la siguiente.

$$p_{Sop} = \left\lceil log_2 \left(f_0' n M_w \frac{2^{N_s}(2^{p_z} - 2)}{2^{N_s} - 2} \right) \right\rceil$$

El procedimiento a realizar para hacer la discretización es el siguiente.

- El diseñador, de acuerdo a la aplicación, debe tener definidos los siguientes parámetros M_z, M_w, p_z, n y p_n.

- Determinar M_s y N_s.

- Obtener el valor óptimo de p_s y m_s.

- En lugar de utilizar todos los N_s bits como índices para acceder a los datos guardados en la LUT, sólo se utilizan los p_s más significativos, por tanto la LUT debe tener 2^{ps}-1 datos, cada una guardando un valor entre M_s y $-M_s$.

Este método puede modificarse, para lograr mayor precisión, por ejemplo utilizando adicionalmente LUTs más pequeñas en los intervalos donde el error puede llegar a ser crítico, esto se acostumbra en los intervalos intermedios de la función de activación sigmoidea.

2.3.3.2 Método por aproximaciones polinómicas. Es un método muy interesante para implementar las funciones de activación de una red neuronal, consiste en combinar polinomios de bajo nivel con pequeñas LUTs. Debido a que, además de grandes capacidades en memoria los FPGAs vienen equipados con munchas unidades aritméticas dedicados (tales como multiplicadores y sumadores) resulta ser una técnica que aprovecha muy bien los recursos. Por otra parte, la interpolación a bajo nivel tiene como ventajas que las mismas estructuras pueden ser usadas para realizar diversas funciones, puesto que lo único que se debe cambiar son los coeficientes polinómicos.

Otro de los aspectos a tener en cuenta en la implementación hardware de redes neuronales es la manera en cómo se lleva a cabo el entrenamiento de la red. Éste implica cálculos aritméticos de elevada complejidad que obligan al empleo de estructuras muy sofisticadas. Éste aspecto, como se ve a continuación en la sección 2.4, puede ser manejado con mucha flexibilidad dentro de un diseño.

2.4 MODOS DE ENTRENAMIENTO DE UNA RED NEURONAL EN FPGA

Este término no debe ser confundido con el tipo de entrenamiento utilizado en redes neuronales (supervisado y no supervisado). Los modos de entrenamiento de una red neuronal están relacionados directamente con el lugar físico donde se lleva a cabo el aprendizaje de la misma. Se puede encontrar dos modos de entrenamiento.

2.4.1 Entrenamiento externo (offline learning). El algoritmo de entrenamiento es un proceso iterativo y como tal puede ser implementado en sistemas secuenciales, tales como las actuales plataformas computacionales (sin que con ello se vea perjudicado el procesamiento en modo de recuerdo) acorde con lo anterior, el aprendizaje de la red en este modo de entrenamiento tiene lugar fuera del FPGA. Cabe destacar que el entrenamiento es un procedimiento que se lleva a cabo hasta obtener los pesos adecuados y posteriormente se desconecta para colocar a la red en modo de recuerdo.
La implementación de este modo de entrenamiento se incluye dentro de las llamadas metodologías de Codiseño Software / Hardware, y posee ventajas que se ven traducidas en ahorro de recursos hardware (CLBs del FPGA) y en flexibilidad, gracias a que se pueden aprovechar todas las ventajas que traen consigo paquetes computacionales especializados en RNA tales como Matlab.

2.4.2 Entrenamiento interno (on-chip learning). Ocurre cuando el algoritmo de aprendizaje es implementado sobre el mismo FPGA. Permite que el sistema sea independiente de cualquier otro sistema digital, pero gasta una cantidad elevada de recursos respecto a sistemas que funcionan con entrenamiento externo.

2.5 BLOQUES BÁSICOS DE UNA RED NEURONAL

Para realizar digitalmente la implementación de una red neuronal en un FPGA se debe primeramente definir una estructura electrónica conformada por un conjunto de bloques básicos que adecuadamente ensamblados conformen el diseño electrónico.

Ocho son los bloques fundamentales que se plantean para la realización de una RNA en un FPGA, los cuales son detallados a continuación.

1. Módulos de Entrada.
2. Unidad de Cómputo Paralelo.
3. Interfaz.

4. Bloque Algorítmico.
5. Sistema de Comunicaciones.
6. Unidad de Almacenamiento.
7. Módulos de Salida**.**
8. Unidad de Control.

2.5.1 Módulos de entrada. Es el encargado de recibir los datos desde el exterior de la red, ya sea desde un arreglo sensorial u otro sistema digital, dependiendo de la naturaleza de la aplicación. Lo ideal para su realización es utilizar estructuras para el traspaso de datos a alta velocidad como por ejemplo el puerto USB pero también pueden utilizarse otras como el puerto paralelo o el puerto RS-232.

2.5.2 Unidad de cómputo paralelo. Es el bloque encargado de realizar todas las operaciones aritméticas de la red neuronal y está constituida por un conjunto de unidades fundamentales trabajando en forma concurrente, denominadas PEs, (tantas de acuerdo al paralelismo que se desea obtener) además de los bloques utilizados para evaluar las funciones de activación. Estos elementos son descritos a continuación.

2.5.2.1 Unidad de procesamiento PE[35]. Es el elemento de la arquitectura encargado de efectuar las operaciones aritméticas con los datos que se le suministren, su estructura depende esencialmente de las tareas que deba ejecutar. Las operaciones que realiza son aquellas que aumentan cuadráticamente con el número de neuronas del modelo, entre este tipo de operaciones se pueden citar las sumas, las restas, los productos y acumulaciones; debido a su frecuencia resultan críticas a la hora de evaluar el tiempo de proceso.

2.5.2.2 Módulo para la función de activación. Es encargado de efectuar las operaciones que no se realizan con tanta frecuencia, pero que resultan computacionalmente complejas, tal es el caso de las funciones de activación neuronales. Como su estructura implica cálculos más complicados se aconseja sea implementado haciendo uso de las unidades dedicadas del FPGA.

2.5.3 Interfaz. Sirve de medio a través del cual otro sistema digital puede transferir datos a la red neuronal. Estos datos son los pesos, Bias (en caso de que la red funcione con modo de entrenamiento externo) y los datos de las LUT para la evaluación de las funciones de activación. Este módulo debe ser capaz de recibir los datos y enviarlos a las locaciones de memorias adecuadas al interior del ANS.

[35] DEL BRÍO y SANZ MOLINA. Op. cit., p. 181.

Clásicamente suele funcionar de dos formas, en modo serial a través del estándar RS232 y en modo paralelo a través del puerto paralelo. Actualmente existen otros medios para el intercambio de datos entre dispositivos digitales que funcionan a más alta velocidad, por ejemplo el puerto USB.

2.5.4 Bloque algorítmico. Se ocupa de todo lo relacionado con el algoritmo de aprendizaje. Dependiendo del modo de entrenamiento en el que funcione la red puede estar o no estar dentro del sistema neuronal. Para más claridad, sólo está presente en redes neuronales con modo de entrenamiento interno (on-chip learning), por lo que en redes con modo de entrenamiento externo (offline learning) éste bloque queda ubicado por fuera del dispositivo.

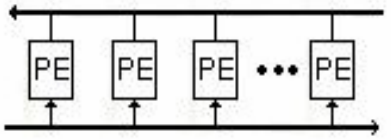
2.5.5 Sistema de comunicaciones[36]. Es el responsable de manejar las comunicaciones internas entre todos los elementos que componen la red neuronal. Entre las funciones que debe realizar están: llevar los datos desde el módulo de entrada hasta la unidad de computo paralelo y desde esta unidad hacia el módulo de salida, ser el medio a través del cual se modifican los pesos y se graban los datos correspondientes en las LUT, y dentro de la unidad de cómputo paralelo, define cómo son conectados los PEs y por tanto la forma en cómo se llevan a cabo los cálculos. El sistema de comunicaciones se caracteriza por una topología de interconexión de buses de comunicación, entre las cuales se distinguen dos arquitecturas diferentes: orientadas a bus y sistólicas (…)[37].

2.5.5.1 Arquitectura orientada a bus o BBA (broadcast bus architecture). Los PEs se conectan entre sí por medio de buses en configuración *broadcast* (figura 14), es decir, todos ellos se encuentran conectados a uno o varios buses generales que los comunican entre sí, y a través de los cuales se transfieren todos los datos, direcciones y operaciones. Habitualmente una configuración del módulo de comunicaciones de este tipo implica el empleo de PEs que incluyen, a diferencia del resto de las arquitecturas, un banco interno de registros para el almacenamiento de los pesos. Es ésta la estructura más utilizada en la implementación de redes neuronales, y una de las más interesantes, debido a su modularidad, versatilidad y fácil escalabilidad, aunque se ve penalizada cuando los pesos sinápticos no caben en la memoria total del PE, lo que obliga a realizar frecuentes intercambios de datos con memorias externas.

[36] Ibíd., p. 184.

37 ANANTH, Grama; ANSHUL, Gupta; KARYPIS, George y KUMAR, Vipin. Introduction to Parallel Computing. England: Pearson Education, 2003.

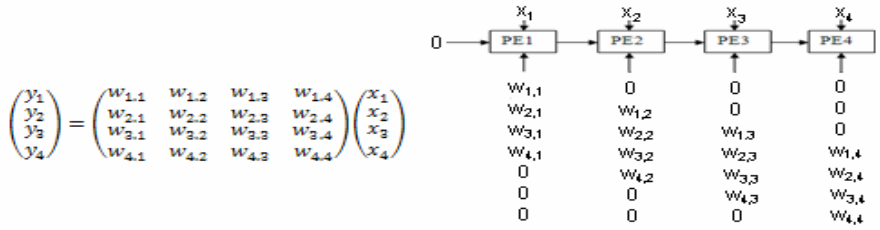
Figura 14. Arquitectura BBA.



Fuente. Redes neuronales y sistemas borrosos.

2.5.5.2 Arquitecturas sistólicas. Se caracterizan porque cada unidad de control se encuentra conectada a unidades de control físicamente cercanas, por tanto si se desean conectar dos PEs lejanos, esta conexión debe ser realizada a través de dos o más PEs. Se denominan arquitecturas sistólicas porque tras un periodo de latencia inicial los resultados de las operaciones van surgiendo de los procesadores en cada ciclo de reloj.

Figura 15. Representación del Flujo de Datos.



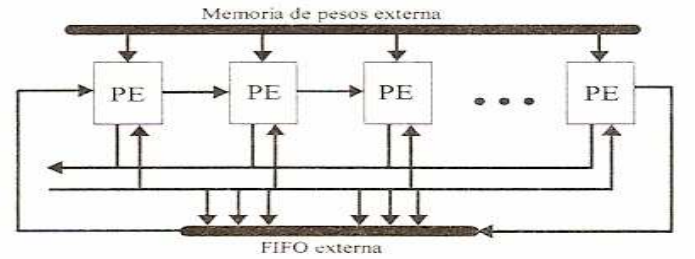
Fuente. Redes neuronales y sistemas borrosos.

Este tipo de arquitecturas de comunicaciones se caracteriza porque todos los PEs se ven implicados en el cálculo de un producto matriz – vector (figura 15). Dentro de estas arquitecturas se tienen:

- **Arquitectura en anillo sistólico o SRA.**

Cada PE se conecta con otros dos formando un anillo de procesadores (figura 16). Los datos con los que opera proceden de dos fuentes, una el procesador anterior en el anillo y otra un bus que conecta los procesadores a la unidad de memoria donde se almacenan los pesos.

Figura 16. Arquitectura sistólica SRA.

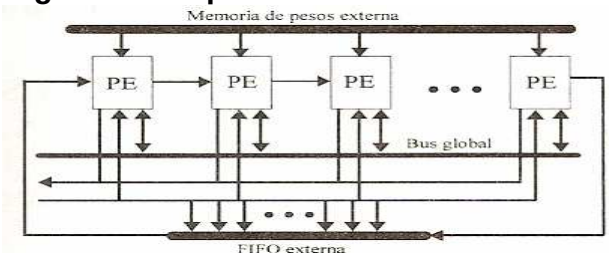


Fuente. Redes neuronales y sistemas borrosos.

- **Arquitectura en anillo sistólico con bus global o SRAGB.**

Se obtiene de la SRA añadiendo un bus global que permite el envió de datos de un PE a cualquier otro en un ciclo de reloj (figura 17). Así se facilita, por ejemplo, la realización de una suma global a partir de las sumas parciales realizadas por diferentes PEs.

Figura 17. Arquitectura sistólica SRAGB.



Fuente. Redes neuronales y sistemas borrosos.

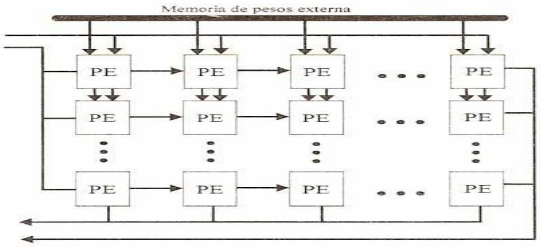
- **Arquitectura en anillo sistólico múltiple o MSRA.**

Se compone de varias SRAs en paralelo, alimentadas con la misma memoria de pesos. Su propósito es obtener una mejora en el rendimiento mediante el procesamiento de varios patrones a la vez.

- **Arquitectura de matriz sistólica cuadrada o SSAA.**

Se trata también de una estructura sistólica cuyos PEs se ordenan en forma de matriz cuadrada (figura 18). Proceden también de una arquitectura clásica de multiplicación de matriz por vector, considerándose como una de las más interesantes al aprovechar muy eficientemente el paralelismo de cálculo.

Figura 18. Arquitectura sistólica SSAA.



Fuente. Redes neuronales y sistemas borrosos.

- **Arquitecturas de anillo sistólico de procesadores o SRMPA.**

Es similar al SRA pero el flujo y manipulación de datos se realiza sobre matrices y no sobre escalares que se construyen concatenando diversos vectores de entrada.

2.5.6 Unidad de almacenamiento[38]. Suministra los datos apropiados en el momento adecuado a la unidad de cómputo paralelo. Involucra dos módulos fundamentales, uno responsable de alojar los valores de pesos y Bias (memoria de pesos) y otro de almacenar resultados intermedios además de las entradas de red (memoria de resultados), ambos módulos pueden adoptar dos estructuras básicas (distribuida y centralizada).

2.5.6.1 Memoria de almacenamiento distribuida. Cada PE dispone de su propia unidad de memoria local, si bien es la configuración ideal cuando la información que cada PE debe elaborar es de origen completamente local, presenta serias restricciones cuando las operaciones que el PE debe realizar implican el uso de datos procedentes de otros procesadores.

2.5.6.2 Memoria de almacenamiento centralizada. Consiste en que una única unidad de almacenamiento proporciona los datos necesarios a todas las unidades de procesamiento, de forma que cada una de ellas tiene acceso a todos los datos.

2.5.7 Módulos de salida. Depende básicamente de la aplicación a realizar o de la función que desempeñe la red neuronal dentro de otro sistema. Puede involucrar funciones tales como la visualización de los resultados entregados por la red, transmisión de señales de control a otros sistemas, señales para monitorización de procesos, entre otras.

38 Ibíd., p. 181.

2.5.8 Unidad de control[39]. Es encargada de controlar a las otras unidades mediante un conjunto de instrucciones y datos. Debe ser diseñada para controlar una arquitectura de computo específica; dicha arquitectura debe ser escogida dependiendo de los requerimientos deseados por el diseñador (tales como paralelismo, estructura de datos, modo de entrenamiento, entre otras) y de la aplicación a desarrollar. De acuerdo con la taxonomía de Flynn, estas arquitecturas se clasifican en cuatro grupos.

2.5.8.1 SISD (single instruction, single data). Son máquinas en las que una instrucción opera sobre un único dato, son las clásicas Von Neumann (Computo Secuencial), en las que se basan la mayor parte de los computadores convencionales, que constan de una única CPU.

2.5.8.2 SIMD (single instruction, multiple data). En ellas, una única instrucción se aplica sobre múltiples datos a la vez. El sistema de control está constituido por una única unidad de control que envía la misma instrucción a múltiples procesadores, que la ejecutaran sobre diferentes datos.

2.5.8.3 MISD (multiple instruction, single data). En estas estructuras un conjunto de instrucciones operan a la vez sobre un único dato. Surgen de manera natural dentro de la taxonomía de Flynn, pero no está claro si tales máquinas existen, aunque se suele citar como ejemplo la técnica de segmentación (*Pipelining*).

2.5.8.4 MIMD (multiple instruction, multiple data). Son máquinas en las que múltiples instrucciones actúan sobre múltiples datos, esta capacidad es lograda debido a que tienen varias unidades de control que envían diferentes instrucciones a distintos elementos de proceso, ejecutándolas con datos locales.

[39] Ibíd., p. 179.

3. ETAPA DE DISEÑO SOFTWARE

Esta etapa en general representa el primer paso en la implementación hardware de redes neuronales, y particularmente en este proyecto marca el comienzo de la parte práctica. Como tal se enfoca hacia el diseño de una red neuronal tipo MLP destinada a cumplir una función específica, la cual es simulada por medio del paquete computacional Matlab.

La herramienta especializada para RNA en Matlab es El Neural Network Toolbox™ 6. La cual permite solucionar un problema de reconocimiento de patrones de tres maneras.

- Mediante el uso de funciones en la *Ventana de Comandos*.
- Mediante la *Interfaz Gráfica de Usuario*, que se despliega con el uso del comando *nntool*.
- Mediante el *Neural Network Pattern Recognition Tool GUI*, que se despliega con el uso del comando *nprtool*[40].

Para el desarrollo de ésta etapa se hace uso del *Neural Network Pattern Recognition Tool GUI*.

3.1 ESTUDIO Y PLANTEAMIENTO DE LA APLICACIÓN

Fueron muchas las aplicaciones estudiadas, entre las que se tienen, reconocimiento de voz, identificación de colores, robot seguidor de línea, reconocimiento de caracteres, determinación de la condición benigna o maligna de tumores en el seno, entre otras.

La escogencia del caso de estudio que se lleva a cabo en este proyecto fue hecha a criterio de los autores, quienes consideran que puede llegar a abrir nuevos campos de investigación en la Universidad de Nariño que contribuyan a la realización de potentes dispositivos en el campo de la electromedicina, razón por la cual se optó elegir la aplicación que consiste en *determinar la condición benigna o maligna de tumores en el seno*.

El cáncer de seno es un crecimiento exagerado del tejido mamario (tumor) y representa una de las principales causas de enfermedad y mortalidad en el mundo, estadísticas afirman que es el 22,8% de todos los cánceres en las mujeres

[40] HOWARD DEMUTH y HAGAN. Op. cit., p. 45.

y se estima que por año se presentan más de 1 millón de casos nuevos[41]. El tumor hallado puede ser benigno, con lo cual se descarta la presencia de cáncer y si es maligno es un indicador de la presencia de esta enfermedad, la cual puede ser curable si se detecta a tiempo.

La finalidad es entrenar una Red Neuronal tipo MLP para clasificar un tumor mamario como *benigno* o *maligno*, basándose en nueve características.

1. Espesor de la Masa.
2. Uniformidad del Tamaño de la Célula.
3. Uniformidad del Estado de la Célula.
4. Adherencia Marginal.
5. Tamaño de la Célula de Epitelio Simple.
6. Núcleo Descubierto.
7. Cromatina Clara.
8. Nucléolo Normal.
9. Mitosis.

Los datos para el entrenamiento de la red son obtenidos de la base de datos de Matlab (cáncer dataset) y son muestras de biopsias tomadas a diferentes pacientes. Se cuenta con 699 muestras (patrones de entrenamiento).

3.2 DISEÑO DE LA RED NEURONAL

El paquete computacional “Matlab” es una herramienta muy eficaz para trabajar con Redes Neuronales Artificiales ya que permite implementarlas, entrenarlas y simularlas de una manera fácil y entrega unos excelentes resultados en un tiempo sumamente corto.

En esta fase del proyecto se debe encontrar la arquitectura de la red que modele en forma correcta el caso de estudio mencionado, ésta es diseñada con el objetivo de implementarse posteriormente en el FPGA; como ya se ha dicho se utilizara un MLP entrenado con el algoritmo Backpropagatión.

Se emplean las nueve características de las muestras de biopsia como entradas de red y la salida se obtiene mediante dos neuronas que indiquen con un uno lógico en la salida de la primera neurona, si el tumor es *Benigno* y con un uno lógico en la salida de la segunda neurona, si el tumor es *Maligno* (tabla 3). El paso siguiente es determinar el número más adecuado de neuronas ocultas.

[41] BOLETIN MANO AL PECHO. Porque el primer paso para detectar el cáncer de seno está en tus manos, Bogotá, 29 de septiembre de 2008. p. 3.

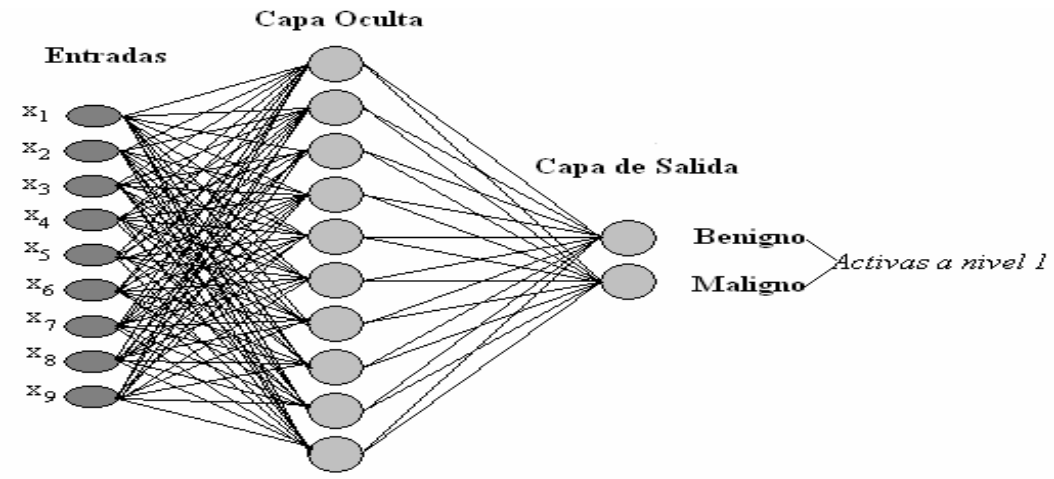
Tabla 3. Clasificación según las salidas de la red.

Salidas		Tipo de Tumor
Neurona 1	**Neurona 2**	
1	0	Benigno
0	1	Maligno
0	0	Error de Clasificación
1	1	Error de Clasificación

No existe una metodología definida para determinar cuántas neuronas son necesarias en las capas ocultas, simplemente se afirma que se debe colocar las suficientes. Para un problema concreto muy bien pudiera ocurrir que el número de neuronas ocultas, para alcanzar una cierta cota de error, sea tan elevado que su implementación sea inabordable por medio de FPGAs. Así la destreza del diseñador se mide en su capacidad para encontrar una arquitectura de red que sea posible de implementar en el dispositivo y que brinde resultados apropiados al problema específico. Debe tenerse en cuenta que no necesariamente un mayor número de neuronas en la capa oculta da una mejor respuesta, en muchos casos sólo trae consigo un gasto elevado de recursos que no aumenta significativamente el rendimiento del sistema.

Tras una serie de pruebas se determinó que un número de neuronas ocultas entre 8 y 12 daban buenos resultados, por lo que finalmente se opto por una arquitectura 9–10–2, es decir con 10 neuronas ocultas (figura 19).

Figura 19. Arquitectura de red.



Otro de los aspectos a tener en cuenta es la escogencia de la función de activación. Los estudios de redes neuronales realizados aportan como regla general que los problemas de reconocimiento de patrones pueden ser resueltos en su mayoría utilizando la función de activación sigmoidea (*Tansig*), tanto en neuronas ocultas como en neuronas de salida. De acuerdo con lo definido en la sección 2.1, el modelo matemático de la red mostrada en la figura 19, queda expresado como:

$$\mathbf{y^2 = Tansig^2(LW^{2,1}Tansig^1(IW^{1,1}x + b^1) + b^2)}$$

3.3 ENTRENAMIENTO SOFTWARE DE LA RED NEURONAL

El objetivo del entrenamiento es determinar las matrices de pesos $\mathbf{LW^{2,1}}$ y $\mathbf{IW^{1,1}}$, además de los vectores $\mathbf{b^1}$ y $\mathbf{b^2}$ que contienen los Bias. Estas matrices y vectores se utilizan en la etapa de diseño hardware (capitulo 4) y representan los parámetros de entrenamiento con los cuales funciona la implementación en hardware reconfigurable del sistema neuronal creado.

El entrenamiento se realiza en Matlab mediante el comando *nprtool*. Los pasos realizados durante esta fase se explican a continuación.

El primer paso es teclear *nprtool* en la ventana de comandos (*Command Widow*), enseguida se abre la herramienta de Redes Neuronales para Reconocimiento de Patrones, posteriormente se debe dar clic en siguiente (next) y aparece la ventana mostrada en la figura 20. Por medio de la pestaña *Load Example Data Set* cargar el set de datos Breast Cancer y luego dar clic en siguiente.

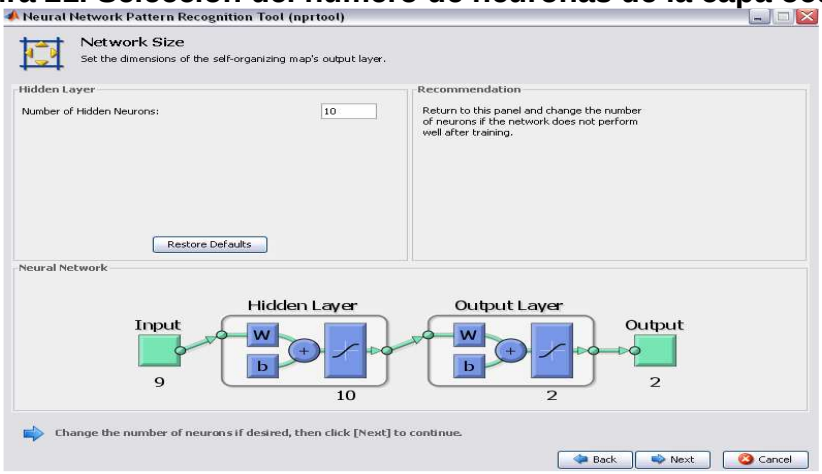
Figura 20. Ventana para cargas datos de entrenamiento.

Fuente. Matlab.

A continuación aparece una opción que permite escoger el porcentaje de datos que serán usados para entrenamiento, validación y prueba (se utilizó el 70%, 15% y 15% respectivamente), colocar los porcentajes deseados y clic en siguiente.

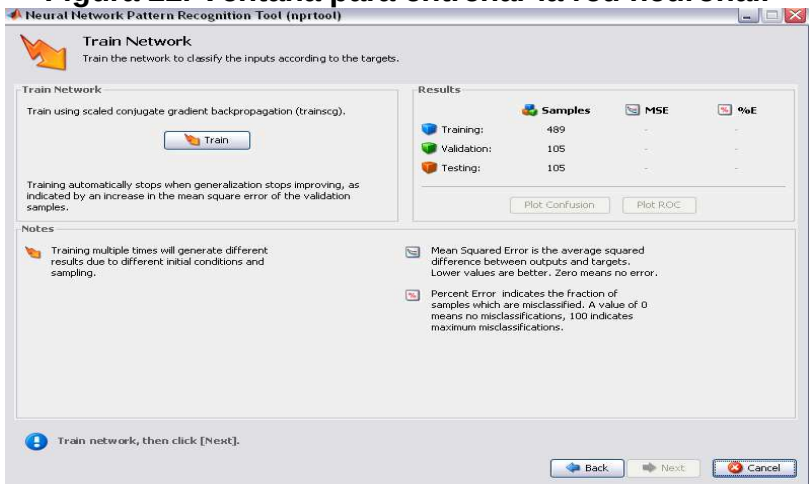
Luego de esto se debe indicar el número de neuronas de la capa oculta (figura 21), como se mencionó anteriormente, para propósitos de este trabajo este número es 10, enseguida hacer clic en siguiente. Posteriormente entrenar la red por medio de la pestaña entrenar (*Train*), como se observa en la figura 22.

Figura 21. Selección del número de neuronas de la capa oculta.



Fuente. Matlab.

Figura 22. Ventana para entrenar la red neuronal.



Fuente. Matlab.

Es importante tener en cuenta que si el entrenamiento no arroja los resultados esperados, la red puede ser re-entrenada hasta alcanzar el desempeño pretendido por el diseñador. Una vez se consiguen resultados óptimos se exportan al Workspace de Matlab, para su posterior análisis.

El entrenamiento de la red se realiza con el Algoritmo Backpropagation de Gradiente Conjugado *trainscg*[42], ya que produce convergencias mucho más rápidas que los demás algoritmos, éste reduce significativamente el número de cálculos por cada iteración, y permite que la red aprenda a realizar tareas de clasificación con un error muy mínimo. La actualización de los pesos se lleva a cabo en modo Batch.

Se debe saber que Matlab usa algunas funciones de pre-procesamiento y post-procesamiento para los datos de entrenamiento, entre éstas se cita *mapminmax* que transforma los datos de entrada de tal manera que tomen valores dentro del intervalo [-1,1], y esta función se aplica por defecto en la red neuronal diseñada.

3.4 ANÁLISIS DE RESULTADOS DE LA ETAPA DE DISEÑO SOFTWARE

Durante el entrenamiento de la red neuronal, Matlab abre una ventana que muestra el progreso de éste (figura 23), entre los parámetros que se indican en dicho proceso están el número de épocas transcurridas (ver sección 1.1.4), el tiempo de entrenamiento, el rendimiento, el gradiente de error y validación cruzada. La validación cruzada permite detener el entrenamiento en el punto óptimo de mínimo error de generalización. Matlab coloca por defecto este parámetro a seis. Para el caso estudiado en este trabajo el entrenamiento se detuvo después de haber transcurrido 34 épocas.

Los 699 patrones de entrenamiento se dividieron en tres grupos, de las cuales 489 son usadas para el entrenamiento, 105 para validación y 105 para prueba. El porcentaje de error obtenido para cada uno de estos grupos de datos es:

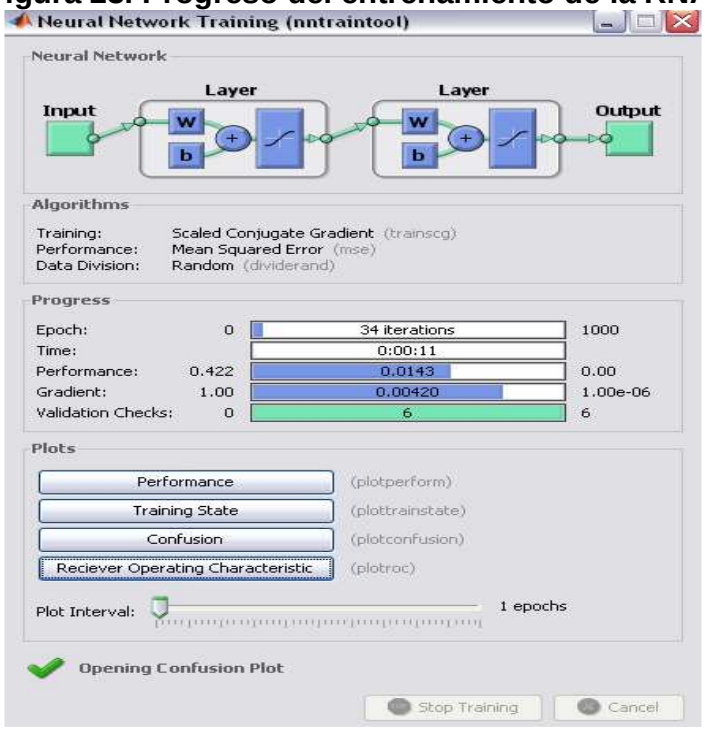
- 1.22199 %, para el conjunto de datos de entrenamiento (error de aprendizaje).
- 2.88461 %, para el conjunto de datos de validación (error de validación).
- 2.88461 %, para el conjunto de datos de prueba (error de generalización).

La gráfica que muestra la evolución del error de aprendizaje, validación y generalización es mostrada en la figura 24. Durante el proceso de entrenamiento, Matlab guarda periódicamente las distintas configuraciones intermedias de pesos

[42] HOWARD DEMUTH, y HAGAN. Op. cit., p. 181.

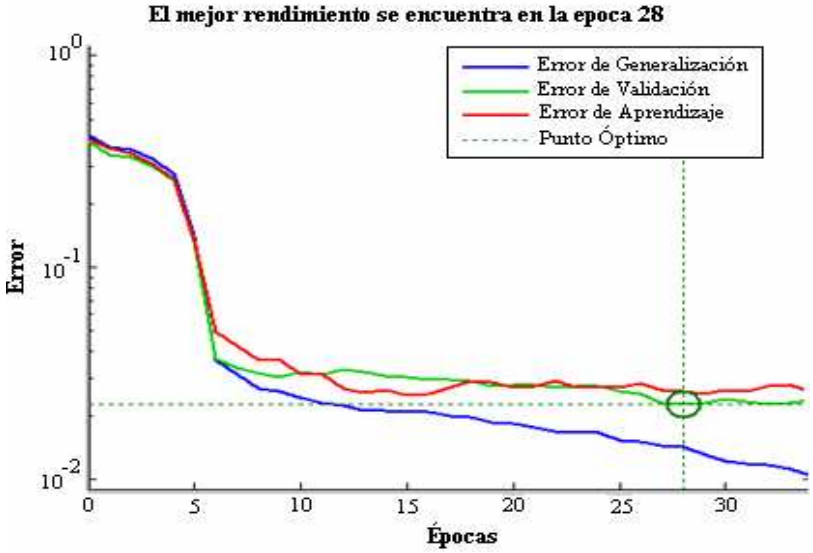
para finalmente quedarse con la que proporciona el mínimo error de validación (punto óptimo). Para la red diseñada el punto óptimo fue obtenido en la época 28.

Figura 23. Progreso del entrenamiento de la RNA.



Fuente. Matlab.

Figura 24. Evolución del error para los 3 conjuntos de datos.



En la figura 25 se muestran los diferentes porcentajes de aciertos y fallas en la clasificación que hace la red para los datos de entrenamiento, validación y prueba. La última matriz (matriz general) muestra estos mismos porcentajes, pero aplicado al total de los datos.

Las diferentes diagonales en color verde, indican los datos que fueron clasificados correctamente. Las diagonales contrarias, en color rojo, indican las muestras que fueron mal clasificadas. La celda en color azul muestra el porcentaje de datos que fueron clasificados de forma correcta (en color verde), y el porcentaje mal clasificado (en color rojo).

De la matriz general se puede concluir que de un total de 699 muestras de entrenamiento, 689 fueron clasificadas correctamente, mientras 10 fueron clasificadas en forma incorrecta, dando como resultado el 98.6% para los datos bien clasificados y el 1.4% para los mal clasificados.

En términos cualitativos se puede decir que la generalización alcanzada por la red neuronal entrenada es muy buena ya que como indican las gráficas, ésta responde muy bien ante la entrada de patrones no utilizados en el entrenamiento, con lo cual el desempeño de la red queda evaluado como muy positivo.

Figura 25. Gráfica de la matriz de confusión.

Fuente. Matlab.

4. ETAPA DE DISEÑO HARDWARE

Los ingenieros durante los últimos tiempos han mostrado una gran fascinación por la velocidad y la eficacia con que las redes neuronales biológicas realizan tareas complejas como el reconocimiento de patrones. Tales redes son capaces de reconocer datos de entrada provenientes de cualquiera de los cinco sentidos con la precisión y velocidad necesaria para permitir que una criatura viviente pueda sobrevivir. Las máquinas que realizan este tipo de tareas con precisiones y velocidades semejantes fueron difíciles de construir hasta los avances tecnológicos de los circuitos VLSI en los años 80. Desde entonces, la implementación con tecnologías VLSI de redes neuronales ha sido testigo de un crecimiento exponencial.

En esta etapa se toma el modelo creado en la etapa de diseño software, y posteriormente, haciendo uso de los conocimientos y experiencias obtenidas en la etapa de investigación, se implementa sobre el FPGA. Por tanto al finalizar esta etapa se obtiene un sistema neuronal implementado en hardware reconfigurable capaz de identificar si un tumor mamario es benigno o maligno.

Las herramientas utilizadas son una tarjeta de desarrollo FPGA y un paquete computacional software de la compañía XILINX.

- **Software.** El software utilizado en este trabajo es la versión web del paquete *ISE 7.1 Xilinx*. Un tutorial básico de esta herramienta se encuentra en el Anexo C. La funcionalidad de esta versión es similar a la versión estándar, pero soporta un limitado número de dispositivos entre los cuales se encuentran los de la familia *Spartan 3.* El software para simulación es *ModelSim XE III de Mentor Graphics*, ambos paquetes son gratuitos y pueden ser descargados del sitio web de Xilix.

- **Tarjeta de Desarrollo FPGA.** Entre las tarjetas que se pueden utilizar se incluyen, *Spartan 3 Starter Kit, Nexy-2 y Basis Boards*, de las cuales todas contienen un dispositivo FPGA Spartan 3/3E y tienen similares puertos de entrada y salida. La tarjeta utilizada en este trabajo es la *Spartan 3 starter Kit* y sus componentes son descritos en el Anexo A.

4.1 DISEÑO VHDL

El enfoque seguido en esta investigación es la derivación efectiva de un sistema neuronal digital y no la sintaxis del lenguaje VHDL, por tanto en lugar de explicar

cada sentencia de lenguaje, se definen un conjunto de prototipos sintetizables, los cuales deben ser integrados para construir cada uno de los bloques del sistema neuronal que posteriormente conformaran el diseño final. Estos prototipos pueden ser fácilmente realizables en VHDL a partir de las especificaciones descritas en la sección 4.1.1. Se invita al lector a profundizar en el diseño digital de circuitos mediante el uso de VHDL, para esto se coloca a disposición la referencia (…)[43].

4.1.1 Prototipos hardware. Son elementos pertenecientes a la arquitectura del sistema neuronal que tienen una función específica dentro de un bloque funcional, ellos pueden ser de tipo combinacional o secuencial. A continuación se describen en forma comportamental cada uno de los prototipos utilizados en el sistema neuronal, los cuales han sido escogidos después de una cuidadosa planeación del sistema total.

4.1.1.1 Prototipos combinacionales. Los Prototipos Combinacionales son circuitos cuyas salidas son función exclusiva del valor de sus entradas en un momento dado, sin que intervengan en ningún caso estados anteriores de las entradas o de las salidas. Las puertas (or,and,nan,xor) son combinacionales donde cada función se puede representar en una tabla de verdad; éstas, carecen de memoria y realimentación.

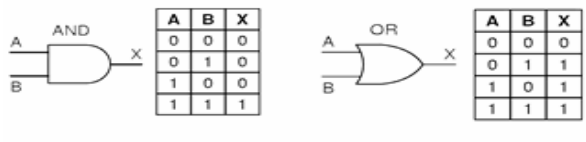
- **Compuerta and.**

Realiza la función lógica *And* convencional, su tabla y su símbolo se muestran en la figura 26.

- **Compuerta or.**

Realiza la función lógica *Or* convencional su tabla y su símbolo son mostrados en la figura 26.

Figura 26. Símbolo y Tabla de Verdad de la Puerta AND y OR.

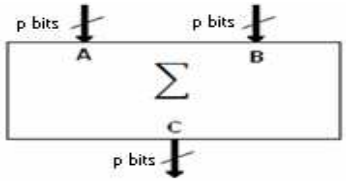


43 BROWN, Stephen y VRANESIC, Zvonko. Fundamentals DIGITAL LOGIC WITH VHDL DESIG. Toronto: Mc Graw Hill, 2004.

- **Sumador con revisión de overflow.**

Este circuito permite sumar dos números con signo de *p* bits cada uno, presentes en los puertos **A** y **B**, los cuales se representan en complemento a 2. La suma obtenida como respuesta, es un número de *p* bits en el puerto **C**, como muestra la figura 27.

Figura 27. Símbolo del sumador con revisión de overflow.

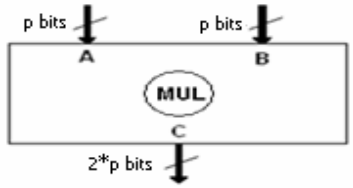


Este sumador tiene como característica, que realiza una revisión contra overflow. Overflow en un sumador de números con signo, ocurre cuando dos operandos que tienen igual signo se suman y se obtiene un resultado de signo contrario; si esto ocurre, el más grande valor (positivo o negativo) debe ser asignado al resultado. Supóngase *p* igual a 8, entonces la adición de dos números positivos debe caer en el intervalo de 0 a 127 mientras que la adición de dos números negativos debe caer entre -128 (el cual es 128 en representación de entero sin signo) y -1 (el cual es 255 en representación de entero sin signo), por ejemplo 65+65=130, en esta situación se presenta overflow, por tanto el sumador debe entregar como resultado el mayor número positivo que se puede representar (127), si no se realiza la revisión contra overflow, el resultado seria -126; por otro lado tomando como ejemplo (-70)+(-70)=-140, para lo cual el sumador debe entregar el mayor valor negativo como respuesta, (-128). Para este caso, sin la revisión contra overflow el resultado seria 116.

- **Multiplicador paralelo.**

Este circuito recibe dos números con signo de *p* bits representados en complemento a 2, por medio de los puertos **A** y **B** y los multiplica. El resultado es un número con signo de 2**p* bits representado en complemento a 2 disponible en el puerto de salida **C.**

Figura 28. Símbolo multiplicador paralelo.



- **Multiplexor.**

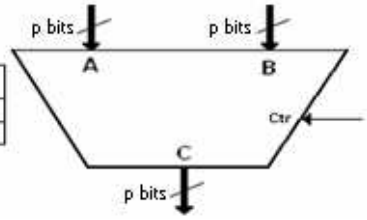
El multiplexor (*MUX*) es un dispositivo que permite dirigir la información digital procedente de diversas fuentes a una única línea de datos, para ser transmitida a través de dicha línea a un destino común. Posee varios puertos de entrada de datos y un único puerto de salida. También posee entradas de selección de datos, que permiten conmutar los datos procedentes de cualquier puerto de entrada hacia el puerto de salida.

El símbolo lógico para un multiplexor de dos entradas se muestra en la figura 29, obsérvese que dispone de una línea para selección de datos *Ctr* que permite que los datos de la entrada seleccionada pasen a la salida.

Si *Ctr* se coloca a nivel bajo se tiene en la salida **C** el dato actual del puerto **A** y si *Ctr* está en nivel alto se obtiene en el puerto de salida **C** el dato actual de puerto **B**.

Figura 29. Símbolo y tabla del mux de dos entradas de datos.

Entrada de selección de datos Ctr	Entrada seleccionada
0	A
1	B



- **Decodificador binario.**

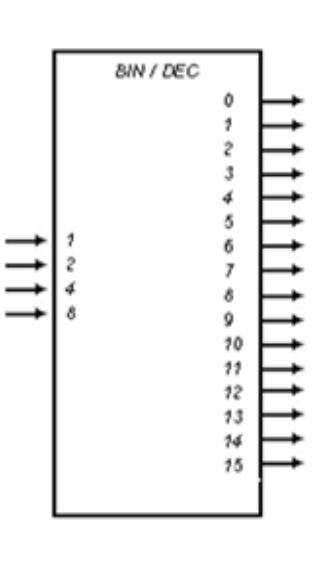
La función básica de un decodificador es detectar la presencia de una determinada combinación de bits (código) en sus entradas y señalar la presencia de este código mediante un cierto nivel de salida, en su forma más general, un codificador posee, líneas para gestionar *g* bits, y en una de sus 2^g líneas de salida indica la presencia de una o más combinaciones de *p* bits. A continuación se describe el decodificador binario de 4 bits, que mas adelante es usado en la conformación de la unidad de cómputo paralelo.

Para poder decodificar todas las posibles combinaciones de 4 bits, se necesitan 16 puertas de codificación (2^4=16), En la figura 30 se muestra una lista de los 16 códigos y sus correspondientes salidas codificadas.

El sistema neuronal desarrollado, requiere únicamente de las 11 primeras líneas de salida de un decodificador de este tipo y por tanto para el diseño y sintetización de este prototipo, las salidas 11, 12, 13, 14 y 15 no se tienen en cuenta, de esta forma el circuito obtenido, posee 4 líneas de entrada y 11 líneas de salida cuyo comportamiento está descrito por la parte no sombreada de la tabla de comportamiento.

Figura 30. Símbolo y tabla de comportamiento para un decodificador binario de 4 bits.

Digito	Entradas binarias A_3 A_2 A_1 A_0	Salidas 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15
0	0000	1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
1	0001	0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
2	0010	0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
3	0011	0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
4	0100	0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0
5	0101	0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
6	0110	0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0
7	0111	0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
8	1000	0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0
9	1001	0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0
10	1010	0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
11	1011	0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0
12	1100	0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0
13	1101	0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0
14	1110	0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
15	1111	0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1



4.1.1.2 Prototipos secuenciales. En estos circuitos, los valores de las salidas, en un momento dado, no dependen exclusivamente de los valores de las entradas en dicho momento, sino también dependen del estado anterior o estado interno.

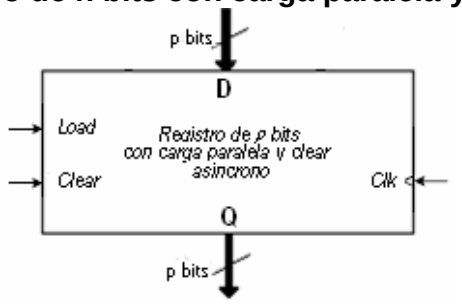
- **Registros de desplazamiento.**

En general estos registros son utilizados para almacenar y desplazar datos que introduce en ellos una fuente externa.

Para ilustrar el modo en que opera un registro de este tipo, se considera la figura 31, en la cual se muestra un registro de *p* bits con carga paralela y clear asíncrono. Los datos de entrada ingresan por el puerto **D**. Cuando la señal de control *Load* se coloca en nivel alto, el dato presente en las líneas de entrada **D** es almacenado en el registro y en el próximo flanco ascendente del reloj *Clk* aparece en el puerto de salida **Q**, cuando *Load* es puesto a nivel bajo la salida **Q** del registro permanece sin cambiar, esta salida puede ser borrada asíncronamente colocando un 1 en la señal de control *Clear*.

En el diseño también son usados otros registros, que difieren únicamente en no poseer un pin para la señal *Clear*.

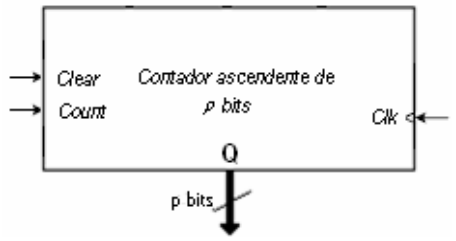
Figura 31. Registro de n bits con carga paralela y clear asíncrono.



- **Contador ascendente.**

Es un circuito capaz de procesar en forma ascendente una secuencia, suponiendo un contador ascendente de *p* bits, éste es capaz de ascender a través de la secuencia (0,1,2,3,4,……,2^p-1). Su símbolo puede apreciarse en la figura 32.

Figura 32. Contador ascendente.



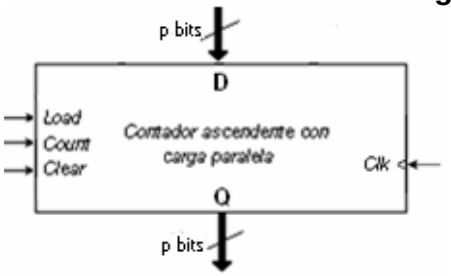
Si la señal de control *Count* es puesta en nivel alto, el contador se incrementara en uno con cada flanco ascendente del reloj *Clk*, esto continuara hasta que la seña *Count* vuelva a estar en nivel bajo. Cuando la cuenta alcance el valor 2^p-1, ésta vuelve nuevamente a ceros. *Clear* es una señal síncrona que al ser puesta en 1 borra la cuenta en el próximo flanco ascendente del reloj *Clk*.

- **Contador ascendente con carga paralela.**

Para hacer un contador binario más versátil es necesario poder inicializar la secuencia de cuenta en cualquier valor diferente de cero. Un contador de *p* bits de este tipo se muestra en la figura 33.

La única diferencia en el funcionamiento de este tipo de contador y el contador ascendente es que éste posee una señal síncrona de control adicional llamada *Load* y un puerto de entrada **D** de *p* bits. Cuando *Load* es puesta a 1 el valor de la cuenta recibe el dato actual en el puerto **D**.

Figura 33. Contador ascendente con carga paralela.

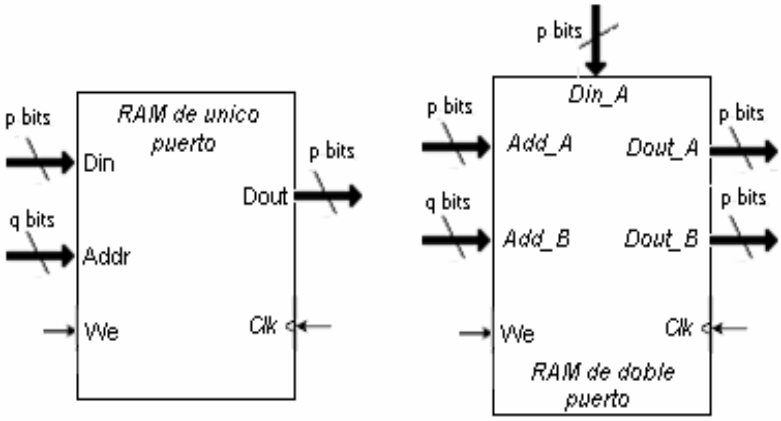


- **Ram de único puerto con lectura síncrona.**

La memoria embebida de los dispositivos de la familia Spartan 3 viene acompañada con una interfaz síncrona, así las operaciones de escritura y lectura son también síncronas.

Si *We* se pone a 1 se realiza la operación de escritura (los datos almacenados se guardan en las posiciones de memoria indicada por las q líneas de dirección del puerto **Addr**), en caso contrario la memoria permanece en modo de lectura indicando en su salida **Dout** el dato guardado en la posición de memoria especificado por las q líneas de dirección, este tipo de memoria se muestra en la figura 34.

Figura 34. Ram de único y doble puerto.



- **Ram de doble puerto con lectura síncrona.**

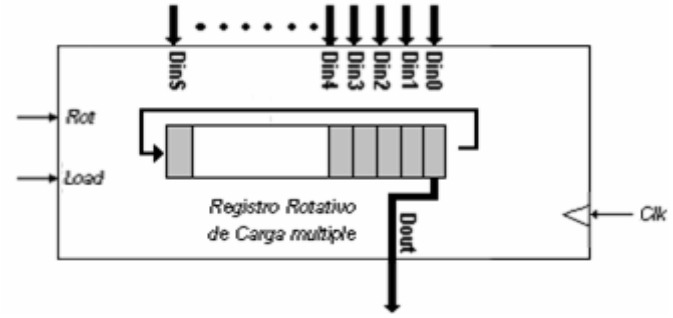
Una RAM de doble puerto incluye un segundo puerto para acceder a la memoria; idealmente, un segundo puerto debería ser capaz de realizar las operaciones de lectura y escritura independientemente y tener sus propias líneas para direccionamiento, entrada de datos y señales de control, sin embargo en este tipo

de memoria, el segundo puerto solamente puede llevar a cabo la operación de lectura. La RAM de doble puerto únicamente se diferencia con la RAM de único puerto, en que la primera posee 2 puertos de dirección **Add_A** y **Add_B** (de q bits cada uno) y dos puertos de salida **Dout_A** y **Dout_B**. La operación de escritura únicamente puede ser realizada por medio de la entrada **Din_A**. Ver figura 34.

- **Registro rotativo de carga múltiple.**

La función para la cual es creado este circuito dentro del sistema neuronal puede compararse con la función de un bus unidireccional, el cual transporta de forma ordenada varios datos procedentes de múltiples unidades de proceso en una sola dirección. Su funcionamiento es tal que, cuando se pone un nivel alto en la línea de control *Load*, los *S* datos (cada uno de p bits) de entrada se cargan concurrentemente en una fila de registros que posee internamente. *Load* es una señal de control síncrona (la carga de datos se ejecuta en el próximo flanco ascendente del reloj *Clk)*. Inmediatamente después de realizarse la carga, el primer dato de entrada **Din0** se hace presente en la salida **Dout**. La señal de control *Rot* (también síncrona) puesta a 1 rota los datos en la fila de registros, con lo cual la salida **Dout** muestra sucesivamente los datos que estaban presentes en los puertos **Din0, Din1, Din2,…, DinS** (cuando se realizo la carga) con cada ciclo de reloj. Este circuito es mostrado en la figura 35.

Figura 35. Diagrama del registro rotativo de carga múltiple.



- **UART.**

La forma más común y sencilla para comunicar cualquier dispositivo con un ordenador es a través de su puerto serie que es compatible con el denominado estándar **RS232** *(o EIA 232 standart)*. En un ordenador puede haber varios puertos series, denominados COM 1, COM 2, etc.

La tarjeta de pruebas S3 tiene un RS232 con el estándar de nueve pines (conector DB9), la tarjeta posee el chip convertidor de voltaje y configura las señales de control automáticamente para empezar la comunicación por puerto serial con un

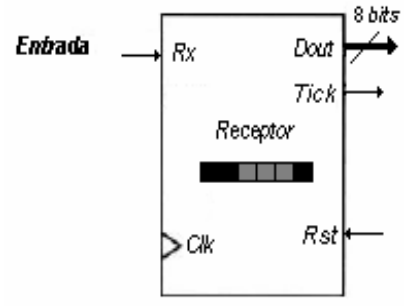
computador, así, el diseñador únicamente debe concentrarse en la realización del circuito UART.
Los parámetros usados en la realización del circuito UART utilizado en este trabajo son los siguientes.

- Velocidad de transmisión de 19.200 baudios
- Un bit de inicio o Start a nivel bajo.
- Dato de 8 bits.
- Corrección de errores sin paridad.
- Dos bits de Parada o Stop a nivel bajo.

El diagrama del sistema UART se observa en la figura 36.

Este circuito obtiene en su salida **Dout** un dato de 8 bits. Los datos entran al sistema en forma serial mediante la entrada *Rx*, la salida *Rx_done_tick* es una señal de estatus que emite un pulso un ciclo después de que haya terminado el proceso de recepción y *Rst* (activo a nivel alto) permite reiniciar la operación del circuito.

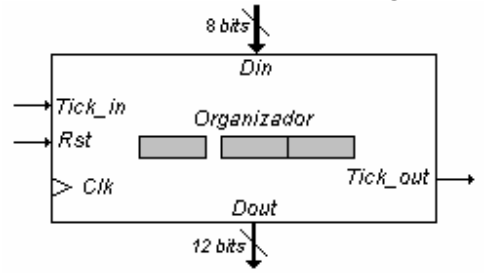
Figura 36. Símbolo circuito UART.



- **Circuito organizador.**

Este circuito es el encargado de dar el formato adecuado a los datos que ingresan al sistema, ya sean pesos, Bias, datos de entrada y valores para la función de activación. Dependiendo de la precisión que se desea obtener, el circuito organizador arma tramas de *p* bits a partir de datos de 8 bits entregados por el receptor UART. Su funcionamiento es el siguiente: supóngase que la trama a realizar sea un dato de 12 bits (*p*=12), donde los cuatro bits más significativos representan las unidades y los 8 bits menos significativos representan la parte decimal, entonces este prototipo debe recibir el primer dato de 8 bits, tomar el nibble menos significativo (unidades), guardarlo en memoria y esperar a que esté listo en la UART el siguiente dato de 8 bits, cuando esto sucede debe tomarlo y armar la trama de 12 bits.

Figura 37. Símbolo circuito organizador.



Tick_in es una señal que informa la disponibilidad de un dato proveniente de la UART y sólo cuando se pone a nivel alto ingresan los datos al circuito. *Tick_out* es una señal de estado que se coloca en nivel alto un ciclo de reloj después de que se haya conformado la trama final. *Rst* puesto a nivel alto reinicia la operación del circuito.

4.2 CARACTERÍSTICAS DEL SISTEMA

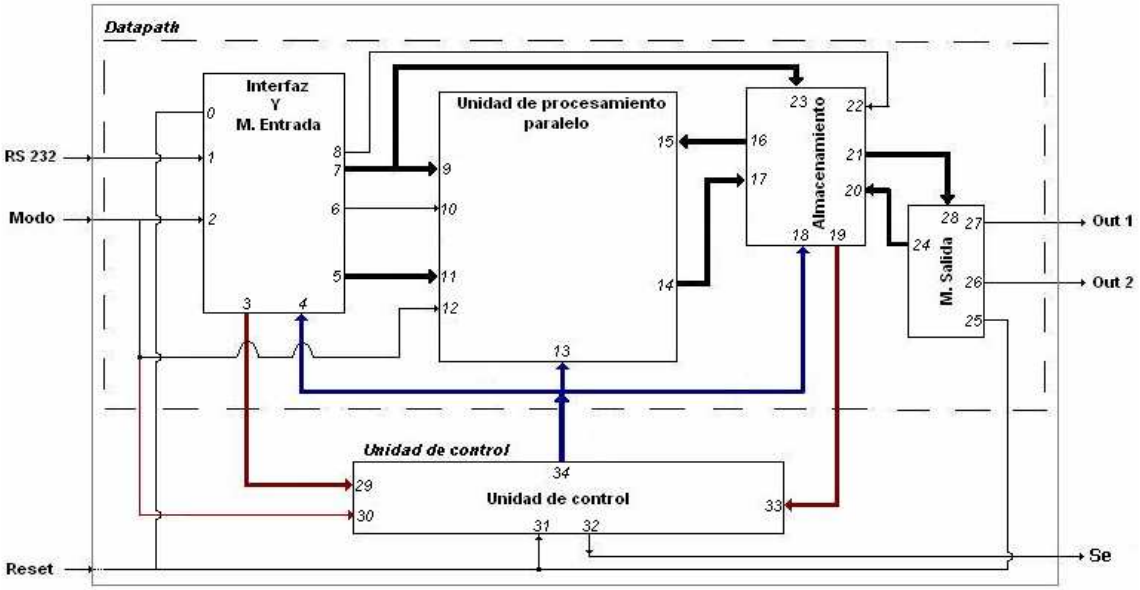
La creación de un sistema neuronal involucra una planeación estricta de cada una de las partes que lo conforman y de cómo estas partes se integran en el sistema final, por tanto requiere de un conjunto de especificaciones para su realización. Definir estas especificaciones fue uno de los objetivos de la etapa investigativa. En función de las especificaciones y requerimientos se puede implementar una gran cantidad de sistemas neuronales artificiales, por tanto es necesario definir de antemano las características del sistema neuronal que se desea crear. A continuación se especifican las características del ANS implementado, las cuales fueron escogidas teniendo en cuenta los recursos disponibles en el FPGA.

- Paralelismo de funcionamiento a nivel sistémico.
- Modo de entrenamiento externo.
- Módulo de entradas en modo serial mediante el estándar RS232.
- Interfaz en modo serial mediante el estándar RS232.
- Unidad de almacenamiento.
  - ✓ Unidad de almacenamiento de pesos distribuida.
  - ✓ Unidad de almacenamiento de datos intermedios centralizada.
- Sistema de comunicaciones orientada a bus BBA.
- Unidad de control tipo SIMD.
- Módulo de salida de estado binario.

4.3 DISEÑO DEL SISTEMA NEURONAL ARTIFICIAL

La metodología aplicada en este punto de la investigación es en modo jerárquico (top down). Como tal, se presenta el sistema total y posteriormente se sigue profundizando en cada una de las partes que lo conforman. La figura 38, muestra como se integran los bloques funcionales del diseño implementado.

Figura 38. Diagrama del ANS implementado.



En la figura 38, se distinguen dos partes principales, denominadas *Datapath* (demarcado con línea punteada) y *Unidad de control*. El *Datapath* puede definirse como una colección de unidades funcionales y es el encargado de procesar todos los datos. La *Unidad de control* es la encargada de generar las instrucciones para controlar el funcionamiento del *Datapath*.

El *Datapath* envía señales de estatus o estado hacia la *Unidad de control* (líneas rojas) y la *Unidad de control* envía señales de control hacia el *Datapath* (líneas azules). Diferenciar entre estas dos señales es de gran importancia en la descripción de cada uno de los bloques funcionales que se lleva a cabo en la sección 4.4.

La figura 38, hace una clara distinción entre pines (puntos a los cuales llegan líneas de conexión delgadas) y puertos (puntos donde llegan los buses o líneas de conexión gruesas). Es muy importante saber que cada bloque funcional, es caracterizado por una palabra de control, que es conformada por todas las líneas

pertenecientes a los buses de control. Estas líneas se distinguen con el nombre del pin del prototipo al cual están conectadas internamente y para evitar confusiones se adhiere un sub-índice con el número del puerto al que pertenecen.

Nótese que la figura 38, en lo que respecta a la unidad de almacenamiento, muestra únicamente la unidad de almacenamiento de datos intermedios, ya que la unidad de almacenamiento de pesos es de tipo distribuida y se encuentra dentro de la unidad de procesamiento paralelo. Otro bloque que no se muestra explícitamente es el sistema de comunicaciones, el cual es conformado por todos los buses de conexión internos y externos de los bloques funcionales.

Por limitaciones de pines en el FPGA, el *módulo de entrada* y la *interfaz* funcionan conjuntamente y hacen uso del mismo puerto RS232, sin embargo, debe aclararse que una aplicación con ANS es mucho más potente si se utilizan, en la realización del módulo de entrada, puertos de alta velocidad para ingreso de datos, tal es el caso del Puerto Paralelo y USB.

4.4 CONFORMACIÓN DE BLOQUES FUNCIONALES

4.4.1 Interfaz y módulo de entrada. Está conformada por el receptor RS-232, el cual recibe los datos por medio del pin 1, cuando éste entrega los 8 bits correspondientes activa la señal *Tick* y el dato pasa al circuito organizador, el cual espera a que este proceso se realice nuevamente para recibir otro dato y así conformar una trama de 12 bits. Cuando este proceso se completa el circuito organizador emite otra señal de salida (*Tick_Out*) a través del pin 8, esta señal es utilizada en todo proceso para el cual es necesario saber cuándo una trama está completa. El pin 2 está directamente conectado al seleccionador de *Modo*, así la compuerta and permite que la señal (*Tick_Out*) esté disponible en el pin *6* únicamente en modo de aprendizaje lo cual sucede cuando se aplica un 1 lógico al seleccionador de *Modo.*

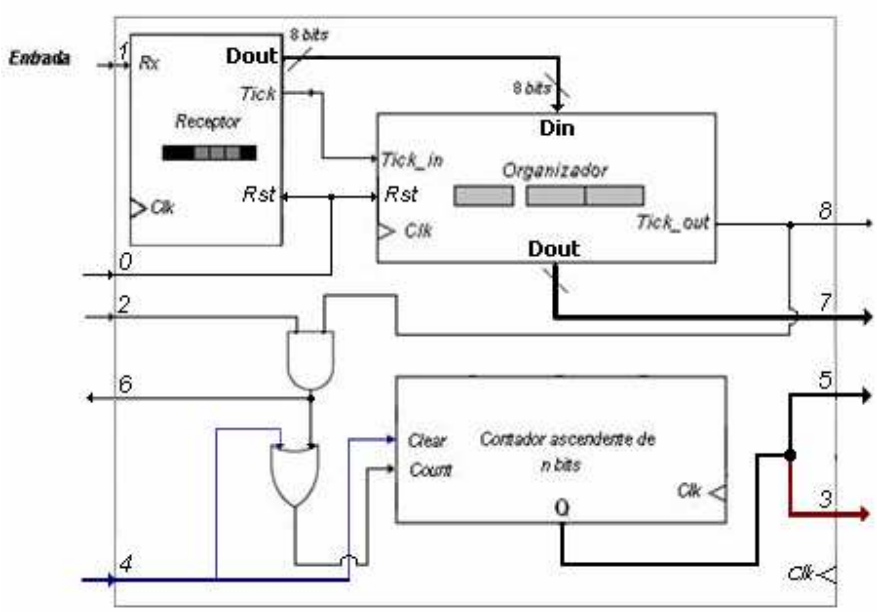
La interfaz posee dos líneas de control que ingresan por medio del puerto de entrada 4, una se conecta a la entrada de la compuerta Or y la otra al *Clear* del contador, de tal forma que la palabra de control que caracteriza a este bloque funcional queda como sigue.

Puerto 4	
$Clear_4$	$Count_4$

El contador cuenta todos los datos que entran al sistema en modo de aprendizaje haciendo uso de la señal *Tick_Out* y permite utilizar estos datos en modo de recuerdo haciendo uso de las líneas del puerto de control 4, para lo cual la

compuerta Or juega un papel importante ya que permite que las señales a través de las líneas de control puedan cumplir su función en su correspondiente modo de funcionamiento.

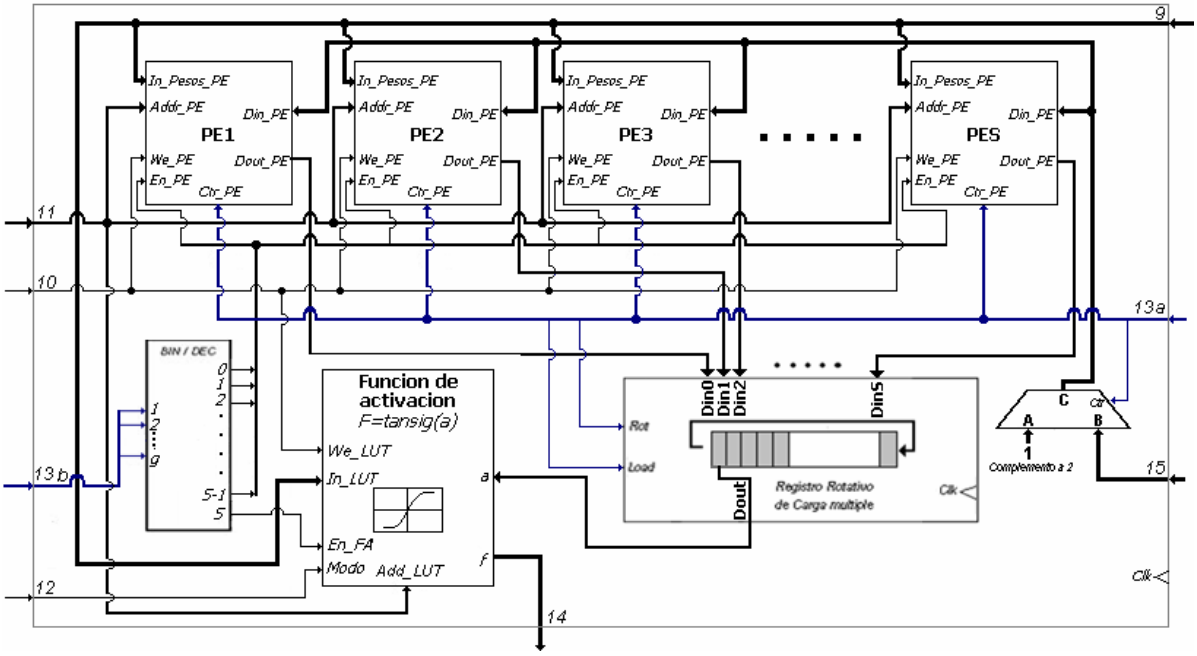
Figura 39. Interfaz y módulo de entrada.



A través del puerto de salida 7 son recibidas las tramas de 12 bits, los puertos 3 y 5 brindan la misma información (estado del contador) pero difieren en que el 3 es utilizado como señal de estado y el 5 es utilizado como direccionador de memoria.

4.4.2 Unidad de cómputo paralelo. Para comprender como se lleva a cabo el procesamiento de los datos en esta unidad se empieza por describir el funcionamiento interno de sus unidades funcionales, los PEs (figura 41) y posteriormente la lógica utilizada en la función de activación (figura 42).

Figura 40. Unidad de cómputo paralelo.



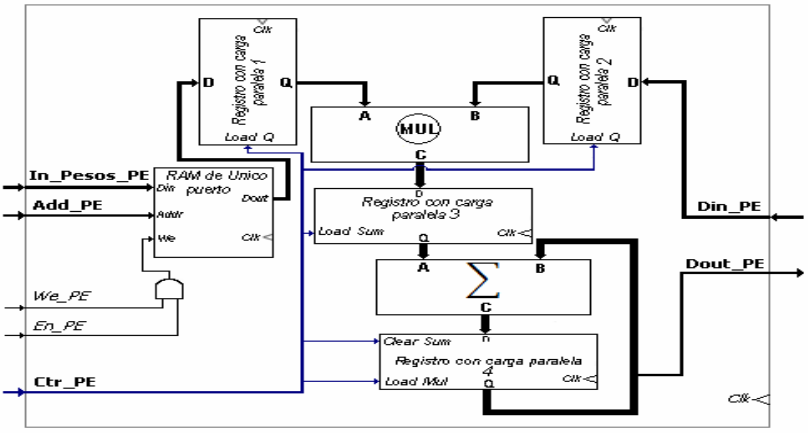
4.4.2.1 Funcionamiento interno de un procesador elemental (PE). El sistema realizado es diseñado para funcionar con un paralelismo sistémico (sección 2.2.4), por tanto cada PE participa en la determinación del valor de salida para una neurona específica y en un tiempo posterior puede utilizarse para determinar la salida de otra neurona en una capa diferente. La suma de productos internos que realizan los PEs se lleva a cabo mediante una capa MAC.

Los datos de entrada en modo de aprendizaje (pesos y Bias) se guardan en la memoria RAM a través del puerto **In_Pesos_PE** en las direcciones especificadas por las líneas **Addr_PE***;* los pines *We_PE* y *En_PE,* ambos puestos a nivel alto, permiten la escritura de datos. El puerto de entrada **Din_PE** es usado como entrada de datos en modo de recuerdo, en el puerto de salida **Dout_PE** se obtiene el argumento de la respectiva neurona, que posteriormente pasara a la lógica correspondiente de la función de activación,

El funcionamiento del PE está controlado mediante el puerto de entrada **Ctr_PE** el cual posee 4 líneas (palabra de control).

Ctr_PE			
LoadQ	LoadMul	LoadSum	ClearSum

Figura 41. Procesador elemental.



Una vez grabados los pesos de entrada y estando en modo de recuerdo se procede a computar el argumento de la función de activación de la siguiente manera.

1. Se debe borrar el contenido del registro 4 mediante la línea de control *ClearSum*.

2. Las líneas **Addr_PE** deben apuntar a la dirección 00_{HEX}, la cual contiene el Bias de la neurona, al mismo tiempo colocar el valor 1 (codificado en complemento a 2) en el puerto de entrada **Din_PE**.

3. Con la línea de control *LoadQ* se carga el dato guardado en la dirección apuntada por **Addr_PE** y el valor presente en la entrada **Din_PE** dentro de los registros 1 y 2 respectivamente, estos valores inmediatamente ingresan al multiplicador.

4. En el siguiente ciclo de reloj el producto obtenido se carga en el registro 3 mediante la línea de control *LoadMul* e inmediatamente este valor pasa a la entrada **A** del sumador para ser sumado con el dato presente en la entrada **B** la cual es el valor actual del registro 4.

5. En el siguiente ciclo del reloj se carga el resultado de la suma en el registro 4 mediante la línea de control *LoadSum*.

6. Incrementar la dirección **Addr_PE** en 1 y a la vez colocar en el puerto **Din_PE** el dato de entrada.

7. Repetir el proceso desde el tercer paso hasta que el dato en **Addr_PE** coincida con el número de entradas a la neurona.

Para un PE que emula una neurona *i* de *n* entradas que pertenece a la primera capa de la red, los datos dentro de la memoria RAM se guardan como se especifica en la tabla 4.

Tabla 4. Almacenamiento de los datos en la memoria ram de un PE.

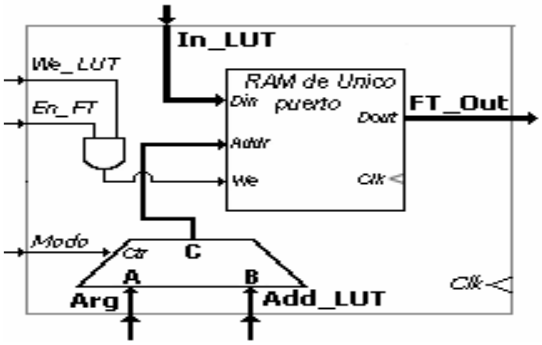
$Direccion_{HEX}$	Dato
00	Bias
01	w_{i1}
02	w_{i2}
. . .	. . .
n	w_{in}

Debe destacarse que cada vez que se cargan en el registro 1 los datos de la tabla 4, en el registro 2 se están cargando las entradas respectivas de la neurona (x_1, x_2,…..,x_n).

El procedimiento descrito se utiliza para emular neuronas de la primera capa, si se desea emular neuronas de otras capas entonces en el segundo paso no se debe apuntar a la dirección 00_{HEX} sino a la dirección en donde se encuentra el Bias correspondiente a la nueva neurona, el proceso restante es análogo al ya explicado.

4.4.2.2 Lógica de la función de activación. La implementación de las funciones de activación en hardware fue discutida en la sección 2.3.3. Para el presente proyecto la función de activación fue desarrollada haciendo uso del método por manejo de tablas.

Figura 42. Función de activación.



El funcionamiento en modo de entrenamiento consiste en grabar en las posiciones adecuadas todos los datos *f(k)* que son usados en el sistema neuronal, donde *f* es la función sigmoidea y *k* es el argumento de la función de activación. En este modo de operación el direccionamiento de la LUT se hace a través de las líneas **Add_LUT** (directamente conectadas al contador de la interfaz), el cual pasa a la salida del multiplexor cuando el pin *Modo* se encuentra en nivel alto.

El funcionamiento en modo de recuerdo se realiza cuando el pin *Modo* está a nivel bajo y consiste en leer todos los datos grabados en modo de entrenamiento y sus direccionamientos se hacen mediante el puerto de entrada **Arg**, (al cual pasan todos los valores de salida provenientes de los PEs).

El puerto **In_LUT** permite el ingreso de los valores en modo de entrenamiento (se conecta directamente al puerto 7 de la interfaz) y el puerto de salida **FT_out** entrega el resultado *f(k)* de la función de activación y es utilizado únicamente en modo de recuerdo.

El pin *En_FA* habilita la escritura de datos en esta unidad y *We_LUT* permite que los datos se escriban cuando la interfaz haya completado toda una trama por tanto está conectado directamente al pin 8 de la interfaz (*Tick_Out*). Esta unidad carece de una línea de control directo ya que es controlada por un decodificador binario (Ver figura 40).

4.4.2.3 Funcionamiento de la unidad de cómputo paralelo. La unidad de cómputo paralelo está conformada principalmente por un conjunto de PEs trabajando concurrentemente (para la RNA implementada en este trabajo se requiere de 10 PEs), la lógica para la función de activación y por un registro rotativo de carga múltiple que permite el flujo de datos para que sean procesados por dicha función.

Nótese en la figura 40 que el puerto de control 13 se divide en dos partes (13a y 13b), los cuales tienen una naturaleza de control distinta y el uso de uno u otro caracteriza el modo de operación en el cual se está trabajando, es importante destacar que el puerto 13a es común a todos los PEs por lo cual en modo de recuerdo todos estos reciben las mismas instrucciones. El puerto 13b toma importancia en modo de entrenamiento y se utiliza para habilitar cualquiera de los PEs en un tiempo determinado, esto permite grabar los pesos y Bias correspondientes a dicho PE, pero además también permite habilitar el modo de escritura en la LUT utilizada en la función de activación. La habilitación de cualquiera de estos circuitos es realizada por medio de un decodificador binario tal como muestra la figura 40. La palabra de control que caracteriza este bloque funcional está dada por:

Puerto 13a							Puerto 13b
$LoadQ_{13}$	$LoadMul_{13}$	$LoadSum_{13}$	$ClearSum_{13}$	$Load_{13}$	Rot_{13}	Ctr_{13}	(Puerto de *g* líneas)

Para poder controlar 10 PEs más la LUT interna en la lógica de la función de activación, el mínimo valor que debe tomar la constante *g* es 4.

Partiendo de que los PEs han procesado un conjunto de datos de entrada completo, entonces los datos resultantes son cargados dentro del registro rotativo de carga múltiple mediante la línea de control $Load_{13}$ y en los siguientes ciclos de reloj pasan ordenadamente hacia la lógica de la función de activación, esto es realizado por medio de la línea de control Rot_{13} del registro rotativo de carga múltiple.

A los procesadores se les debe garantizar un valor de 1 codificado en complemento a 2 (entrada del Bias), esto se realiza mediante el multiplexor, colocando a nivel bajo la línea de control Ctr_{13}. Cuando esta línea de control se encuentra a nivel alto los valores que pasan a través del multiplexor son las entradas de la red ó los datos intermedios.

4.4.2.4 Aspectos generales de la representación de datos. El sistema de representación utilizado fue en punto fijo con complemento a 2. Haciendo un análisis minucioso de la estructura y los recursos del FPGA se determinó que el *mínimo rango de precisión permitido* eran 12 bits. En el presente trabajo se optó por utilizar 4 bits para la parte entera y 8 bits para la parte decimal tanto para pesos como para entradas, por tanto a la salida de los multiplicadores se tienen datos de 24 bits de los cuales 8 son parte entera y 16 parte decimal, esta última representación se mantiene a la salida del sumador, pero en la salida de cada PE se debe recortar a 10 bits, que corresponden al bit de signo, a los tres bits menos significativos de la parte entera y a los seis bits más significativos de la parte decimal. Los 10 bits satisfacen la condición de que todos los valores necesarios para la función de activación estén presentes en la LUT, lo cual se denomina el valor óptimo p_{sop} de acuerdo con lo descrito en la sección 2.3.3.1.

En la etapa software se entrenó una red neuronal para identificar si un tumor mamario era benigno o maligno, con lo cual se obtuvieron dos matrices de pesos ($\mathbf{LW}^{2,1}$ y $\mathbf{IW}^{1,1}$) y dos vectores ($\mathbf{b}^1$ y $\mathbf{b}^2$) que contienen los Bias. Teniendo definidas dichas matrices y vectores, uno de los propósitos de esta etapa es enviar estos datos hacia el diseño creado en el FPGA para lo cual se utilizan tres programas creados en Matlab, que permiten además de enviar los pesos y Bias, enviar también los datos de la LUT (función de activación). Estos tres programas se encargan respectivamente de convertir los datos del entrenamiento a complemento a 2, ordenarlos y enviarlos. Estos programas son utilizados tanto en

modo de aprendizaje (envío de datos de entrenamiento) como en modo de recuerdo (envió de entradas).

4.4.3 Sistema de comunicaciones. Este bloque queda implícitamente definido dentro del sistema completo. Por la manera en cómo están conectados los PEs queda incluido dentro de las *Arquitecturas Orientadas a Bus* (*BBA*). Todos los PEs se comunican mediante un dispositivo que hace las veces de un bus unidireccional que transporta los datos de forma ordenada hacia otra unidad funcional y cada PE cuenta con un sistema de registros para el almacenamiento de pesos (memoria de pesos distribuida).

4.4.4 Unidad de almacenamiento. Como se discutió en la sección 2.5.6 esta unidad está compuesta por dos módulos, uno para el almacenamiento de datos intermedios y entradas y el otro para los pesos y Bias de las neuronas.

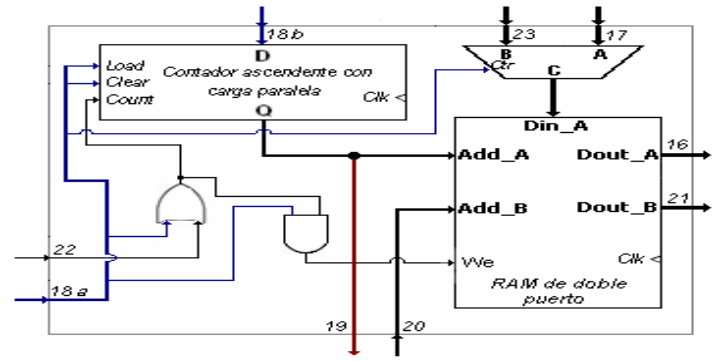
La memoria de pesos es de tipo distribuida y está incluida dentro de la *Unidad de Cómputo Paralelo* por lo cual en este apartado se pasa a describir únicamente el funcionamiento de la memoria de datos intermedios (figura 43).

La palabra de control utilizada para llevar a cabo el proceso que realiza esta unidad se transmite a través del puerto 18, el cual está compuesto por dos partes a y b, como se observa a continuación.

Puerto 18a					**Puerto 18b**
$Count_{18}$	$Clear_{18}$	$Load_{18}$	Alm_{18}	Ctr_{18}	D

La función de esta unidad inicia con el almacenamiento de las entradas de la red a través del puerto 23, para esto la línea de control Ctr_{18} del Mux debe estar a nivel bajo, en caso contrario se guarda en la memoria los datos provenientes de la unidad de cómputo paralelo transferidos por medio del puerto 23.

Figura 43. Unidad de almacenamiento de datos intermedios.



A diferencia de los anteriores bloques éste únicamente funciona en modo de recuerdo, el direccionamiento de la memoria se realiza por medio del contador ascendente con carga paralela, el cual se incrementa en 1 cada vez que una trama se completa en la interfaz, esta información es brindada por la señal *Tick_Out (pin 8)* desde la interfaz que se conecta al pin 22 de este bloque. Cuando todas las entradas son guardadas, éstas se pasan ordenadamente a la unidad de cómputo paralelo por medio del puerto 16, para lo cual se debe borrar el contador (con la línea de control $Clear_{18}$) y enseguida incrementarlo por medio de la línea de control $Count_{18}$; posteriormente entra en un estado de espera hasta que los PEs procesen y envíen los resultados, que como ya se dijo entran por el puerto 23, y son grabados nuevamente en las posiciones siguientes de la memoria; a continuación mediante el puerto de control 18b, en conjunto con la línea de control $Load_{18}$, se carga el dato D que corresponde a la dirección donde se comienza a guardar los datos intermedios y el proceso se repite en forma análoga un número de veces proporcional al número de capas de la RNA. Este bloque tiene internamente una memoria RAM de doble puerto con el propósito de poder acceder en cualquier momento a cada uno de los resultados intermedios (incluyendo la salida de la red) para un determinado patrón de entrada. Estos resultados son obtenidos a través del puerto 21 y se direccionan haciendo uso del puerto 20. El puerto 19 brinda una señal de estatus que informa el estado del contador interno.

4.4.5 Unidad de control. Al inicio de esta etapa se definieron dos partes fundamentales del sistema, las cuales se nombraron como Datapath y Unidad de Control. El Datapath está conformado por todas las unidades anteriormente descritas, éste al igual también tiene una palabra de control que corresponde a todas las palabras de control unidas. Estas pueden ser escritas en un orden arbitrario, pero una vez definido el orden se debe ser consistente con éste, en todo lo que sigue del diseño.

La unidad de control se encarga de generar las señales apropiadas con las cuales se controla el Datapath. Esta es básicamente una máquina de estado finito que emite una instrucción en cada estado dependiendo de las señales de estatus que llegan a ella; las instrucciones son comunicadas al Datapath mediante las palabras de control que caracterizan a cada bloque funcional.

Para el diseño realizado fue necesario hacer uso de 24 instrucciones, divididas en 12 instrucciones de control estáticas (tabla 5) y 12 de control dinámicas (tabla 6).

Las instrucciones de control estáticas comandan el funcionamiento del sistema en modo de recuerdo, además mantienen el flujo en el estado *entrenamiento* mientras se guardan los pesos, Bias y datos en la LUT, éstas no cambian mientras se está ejecutando un estado especifico, por lo cual cada estado tiene su propia instrucción de control estático.

Las instrucciones de control dinámico gobiernan el funcionamiento del sistema en modo de aprendizaje, por tanto pueden ejecutarse todas dentro de un mismo estado, dicho estado es siempre el de *entrenamiento*.

Tabla 5. Instrucciones de control estáticas.

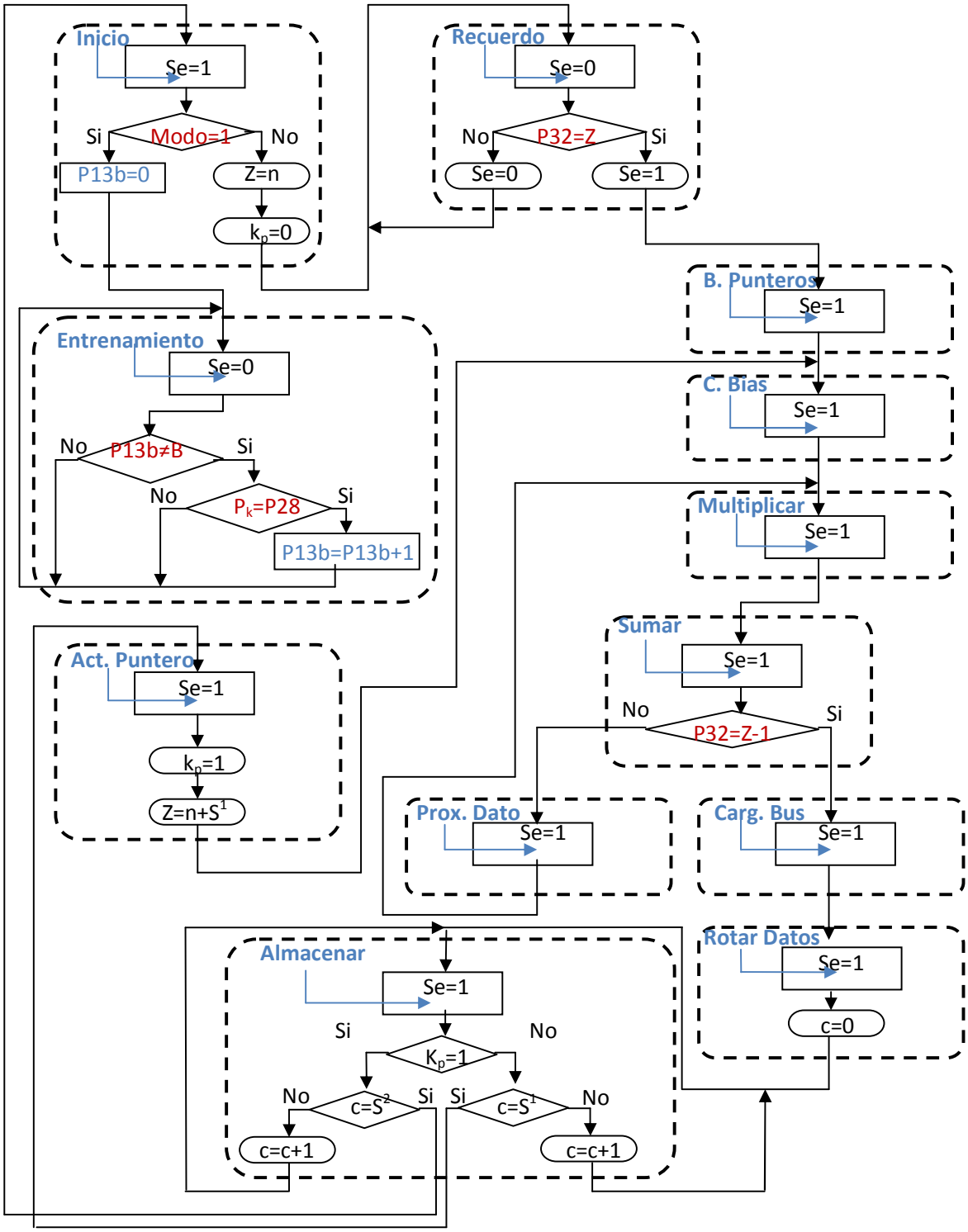
Nº	Instrucción	Puerto 4		Puerto 13a							Puerto 18a					Puerto 18b
		$Clear_4$	$Count_4$	$LoadQ_{13}$	$LoadMul_{13}$	$LoadSum_{13}$	$ClearSum_{13}$	Load	Rot	Ctr	$Count_{18}$	$Clear_{18}$	$Load_{18}$	Alm_{18}	Ctr_{18}	D
1	Inicio	1	0	1	0	0	1	0	0	0	0	1	0	0	0	0000000000
2	Recuerdo	0	0	0	0	0	0	0	0	0	0	0	0	1	1	0000000000
3	Entrenamiento	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0000000000
4	Borrar Punteros	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0000000000
5	Cargar Bias	0	1	1	0	0	1	0	0	1	0	0	0	0	0	0000000000
6	Multiplicar	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0000000000
7	Sumar	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0000000000
8	Próximo Dato	0	1	1	0	0	0	0	0	0	1	0	0	0	0	0000000000
9	Cargar Bus	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0000000000
10	Rotar Datos	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0000000000
11	Almacenar	0	0	0	0	0	0	0	1	0	1	0	0	1	0	0000000000
12	Actualizar Puntero	0	0	0	0	0	0	0	0	0	0	0	1	0	0	Dirección

Tabla 6. Instrucciones de control dinámicas.

Nº	Instrucción	Puerto 13b			
		$13b_4$	$13b_3$	$13b_2$	$13b_1$
1	Habilitar Escritura PE1	0	0	0	1
2	Habilitar Escritura PE2	0	0	1	0
3	Habilitar Escritura PE3	0	0	1	1
4	Habilitar Escritura PE4	0	1	0	0
5	Habilitar Escritura PE5	0	1	0	1
6	Habilitar Escritura PE6	0	1	1	0
7	Habilitar Escritura PE7	0	1	1	1
8	Habilitar Escritura PE8	1	0	0	0
9	Habilitar Escritura PE9	1	0	0	1
10	Habilitar Escritura PE10	1	0	1	0
11	Habilitar Escritura LUT	1	0	1	1
12	Deshabilitar Escritura PEs	0	0	0	0

La progresión de estados, a través de los cuales avanza la unidad de control, cuando sobre ella se aplica una señal de reloj, Se describen a continuación mediante el diagrama de flujo de estados (figura 44),

Figura 44. Diagrama de flujo de la unidad de control.



Las líneas punteadas en el diagrama de flujo encierran todas las operaciones que realiza la unidad de control en cada uno de los estados, las flechas azules entrantes a los rectángulos indicadores significan que en ese punto debe ejecutarse la respectiva instrucción de acuerdo con la tabla 5.

La señal *Se* físicamente corresponde al pin 32 e informa la disponibilidad del sistema para recibir un conjunto de entradas. Por tanto se recibe un nuevo conjunto de entradas únicamente cuando *Se* esté a nivel bajo; en caso contrario significa que el sistema está realizando cálculos internos.

La unidad de control posee un pin de *Reset* (físicamente corresponde al pin 31) que permite reiniciar la operación del sistema en cualquier momento.

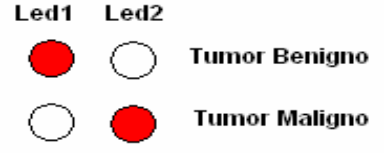
En el diagrama de flujo se hace uso de una serie de señales, variables y constantes adicionales que se explican a continuación:

- n representa el número de entradas a la neurona.
- Z se usa como puntero de entradas.
- k_p se usa como puntero de capas.
- S^1 y S^2 representa el número de neuronas en las capas 1 y 2 respectivamente.
- c se usa como puntero de neuronas.
- P_k es una constante definida como $P_k = S^1 + S^2 +$ # de capas.
- Modo, seleccionador del modo de operación (ver figura 40).
- P33, señal de estado que ingresa por el puerto 33 (ver figura 40).
- B, número de PEs (en el diseño realizado toma un valor de 10).
- P13b es una señal de control que permite habilitar un PE específico e ingresa a la unidad de cómputo paralelo por el puerto 13 (ver figura 42). Internamente en la unidad de control también esta señal también se usa como señal de estado.
- P29, señal de estado que ingresa por el puerto 29 (ver figura 40).

4.4.6 Módulo de salida. Depende exclusivamente de la aplicación para la cual fue diseñada la red neuronal. Dado que el objetivo de la red neuronal entrenada en esta investigación es informar el estado benigno o maligno de un tumor de seno, se optó por presentar los resultados en forma visual para lo cual se usan dos LEDs. El funcionamiento del módulo de salida consiste básicamente en acceder a los datos guardados en el módulo de almacenamiento de datos intermedios en las direcciones donde se guardan los resultados finales y entonces hacer encender un LED en función de un circuito de decisión que determina si un dato en complemento a 2 tiende a 1 (tumor benigno), ó a -1 (tumor maligno).

La información recibida se interpreta como indica la figura 45.

Figura 45. Visualización de resultados.



4.5 ANÁLISIS DE RESULTADOS DE LA ETAPA DE DISEÑO HARDWARE

El desempeño de una red neuronal realizada en hardware o en software puede medirse en relación a la generalización, que es producto del entrenamiento de la red. Debido a que la red implementada funciona con modo de entrenamiento externo, la generalización alcanzada depende exclusivamente del entrenamiento realizado en la etapa software, así de acuerdo con el estudio desarrollado en dicha etapa se encuentra que el MLP propuesto entrenado con Backpropagation es capaz de determinar correctamente la condición benigna o maligna de un tumor mamario en el 98.6% de las ocasiones.

La tabla 7, muestra el reporte final del proceso de síntesis para el diseño desarrollado. Ella indica la cantidad de recursos hardware utilizados.

Tabla 7. Reporte final del proceso de síntesis.

```
=========================================================================
*                            Final Report                               *
=========================================================================

Device utilization summary:
---------------------------

Selected Device : 3s200ft256-4

 Number of Slices:                      784  out of   1920    40%
 Number of Slice Flip Flops:            869  out of   3840    22%
 Number of 4 input LUTs:               1120  out of   3840    29%
 Number of bonded IOBs:                  22  out of    173    12%
 Number of BRAMs:                        12  out of     12   100%
 Number of MULT18X18s:                   10  out of     12    83%
 Number of GCLKs:                         1  out of      8    12%


=========================================================================
```

Fuente. Ise web pack 7.1

El reloj interno que trae la tarjeta de desarrollo S3 es de 50 MHz, pero el reporte del proceso de síntesis informa que se puede trabajar a una frecuencia máxima 126.614 MHz. Para ello, la tarjeta trae consigo una base para un cristal oscilador auxiliar (Ver Anexo A).

4.5.1 Análisis del desempeño de una red neuronal desarrollada en software con respecto a otra implementada en hardware reconfigurable. El poder de procesamiento de las redes neuronales se debe exclusivamente al paralelismo que manejan. Dicho paralelismo debe ser utilizado con el propósito de aumentar el rendimiento de la red, así la decisión de realizar una aplicación hardware o software debe tomarse en relación al rendimiento que se desea alcanzar y de la aplicación a implementar.

Respecto a las RNA implementadas en software se puede decir que su verdadero corazón es el microprocesador, que actúa ejecutando en serie una secuencia de instrucciones o programa (software). Por tanto, la potencia de este tipo de implementaciones depende directamente de la frecuencia de operación, la cual entre mayor sea, permite ejecutar un mayor número de instrucciones, logrando así un menor tiempo de procesamiento. Estas implementaciones tienen a su favor una evolución permanente encaminada a diseñar microprocesadores cada vez más veloces.

Por otro lado, las implementaciones en hardware reconfigurable no operan en base a un microprocesador, sino que su poder de procesamiento se debe a varias unidades (PEs) o neuronas ampliamente interconectadas conformando una red. Estas unidades en realidad son un pequeño procesador, sencillo y lento (en comparación con los actuales microprocesadores). Sin embargo, en este tipo de implementaciones puede utilizarse varios de ellos trabajando en paralelo, esto permite ejecutar múltiples instrucciones en un ciclo de reloj, con lo cual se disminuye el tiempo de procesamiento. Por sus características una implementación de redes neuronales en hardware reconfigurable resulta ser muy costosa en términos de recursos (CLBs del FPGA), debido a la masiva interconexión que poseen, lo cual hace algunos años las hacía poco aplicables en problemas concretos, pero con la evolución de la microelectrónica y las tecnologías VLSI, esta dificultad está siendo superada (Ver Tabla 2).

Por último, se debe decir que el uso de tecnologías reconfigurables trae consigo mismo una ventaja adicional respecto a los microprocesadores, esto es, que gracias a su flexibilidad, se pueden crear unidades de proceso y de control dedicadas, las cuales son controladas mediante un conjunto de instrucciones específicas para un problema en particular y que pueden ejecutarse en menos ciclos de reloj respecto a los microprocesadores de propósito general. Para el caso del sistema implementado en esta investigación, todas las instrucciones se ejecutan en un único ciclo de reloj y sobre diferentes datos a la vez, lo cual es una gran ventaja en comparación con los microprocesadores que requieren comúnmente de cuatro ciclos de reloj para ejecutar la mayoría de las instrucciones.

La tabla 8, compara las principales características del diseño realizado en hardware reconfigurable y la red entrenada en software.

Tabla 8. Comparación hardware reconfigurable frente a software.

	Hardware	Software
Frecuencia de Operación	50 MHz - 126.614 MHz	Varios GHz
Estilo de Procesamiento	Paralelo	Secuencial
Número de Procesadores	10 PEs	1
Conexiones	Muchas	Pocas
Almacenamiento del Conocimiento	Distribuido	Direcciones Fijas
Tolerancia a Fallos	Amplia	Nula
Ciclos por instrucción	1	4 - 8

APORTE A FUTUROS TRABAJOS

Las Redes Neuronales Artificiales han evolucionado ampliamente en los últimos años, por lo cual una gran cantidad de personas procedentes de muy diversa áreas tales como la electrónica, física, matemáticas, ingeniería, biología y psicología hacen uso de ellas en sus investigaciones. Es precisamente dentro de este contexto donde el presente trabajo hace su aporte, dadas las grandes posibilidades que representa el desarrollo de aplicaciones neuronales en dispositivos de lógica programable y que emergen como una alternativa para brindar soluciones eficaces a distintos problemas tecnológicos en la región. Sin embargo debe tenerse presente que el empleo de redes neuronales persigue un objetivo mucho más modesto que la creación de un cerebro artificial, pero son muy potentes para la resolución de problemas prácticos concretos donde se precisa trabajar con muchos datos, en entornos que incluyan mucho ruido, y que además se cuente con un conjunto amplio de datos o casos históricos, como puede ser el caso del reconocimiento de vehículos en los peajes de las autopistas ó la previsión del consumo eléctrico, entre muchos más.

Se quiere dejar en claro que aunque las redes neuronales permiten resolver muchos problemas, pueden no ser la mejor solución en algunos casos (por ejemplo cuando hay un algoritmo que soluciona eficazmente el problema), por tanto siempre es necesario, primeramente realizar una descripción detallada del problema con el propósito de averiguar si algún aspecto de él puede ser resuelto mediante un módulo neuronal o si resulta más conveniente aplicar técnicas convencionales. Lo más aconsejable es separar el problema en partes y resolver cada aspecto mediante el módulo que resulte más adecuado. Acorde con esto las redes neuronales implementadas en hardware reconfigurable representan una herramienta poderosa a añadir a las ya existentes (sistemas digitales, inteligencia artificial, reconocimiento de patrones, estadística, procesamiento de señales, etc.).

CONCLUSIONES

El poder de procesamiento de las redes neuronales se debe exclusivamente al paralelismo innato que manejan. Esta característica les permite solucionar con eficacia problemas en los que la información se presenta masiva, imprecisa y distorsionada y en los cuales los sistemas digitales convencionales parecen no tener mucho éxito.

Una implementación de redes neuronales en hardware reconfigurable debe soportar internamente un gran número de conexiones y un elevado flujo de comunicación de datos. Esto conduce a la utilización de una elevada cantidad de recursos hardware en el dispositivo que puede limitar el tamaño de la red. Dicha limitación está siendo superada gracias a los permanentes avances en tecnología microelectrónica que permiten integrar millones de puertas en un pequeño chip.

Por sus características, los FPGAs resultan atractivos para la implementación de redes neuronales, éstos son capaces de brindar el paralelismo inherente a dichas redes, vienen dotados de altas capacidades en memoria y con bloques aritméticos dedicados; además su reconfigurabilidad se puede aprovechar para optimizar la evaluación de funciones mediante la creación de tablas de búsqueda internas LUTs y también para la creación de estructuras controladas por instrucciones que requieren menos ciclos de reloj.

Para un problema específico, no necesariamente un mayor número de neuronas da una mejor respuesta, en muchos casos sólo trae consigo un gasto elevado de recursos que no aumenta significativamente el rendimiento del sistema. Así la destreza del diseñador se mide por su capacidad para encontrar una arquitectura de red que brinde resultados apropiados con el menor número de neuronas.

Crear un sistema neuronal involucra la planeación estricta de cada una de las partes que lo conforman y de cómo estas partes se integran en el sistema final, por tanto requiere de un conjunto de especificaciones para su realización. La variación de estas especificaciones conduce a implementar sistemas neuronales artificiales con diferentes prestaciones (paralelismo, flexibilidad, precisión, velocidad, etc.). El aumento de dichas prestaciones supone una mayor utilización de área de silicio. Así, la escogencia de las estructuras a utilizarse (bloques funcionales) en un diseño se debe realizar asumiendo que éstas deben alcanzar un balance adecuado entre prestaciones y costos (en términos de CLBs del FPGA).

RECOMENDACIONES

Utilizar un protocolo de alta velocidad para entrada de datos, para lo cual se debe hacer uso de un FPGA mucho más potente que soporte más altas frecuencias y que no tenga limitaciones en el número de pines.

Programar el algoritmo de entrenamiento en hardware (modo de entrenamiento interno) y así lograr que la red diseñada sea independiente del PC, para lo cual se recomienda diseñar una unidad de control tipo MIMD con el propósito de generar instrucciones de control diferentes tanto para el *datapath* como para el *algoritmo de entrenamiento*, las cuales puedan ejecutarse al mismo tiempo.

El rendimiento del sistema se puede mejorar ascendiendo en el nivel de paralelismo de la red, lo cual requiere de un FPGA con muchos más recursos.

Es posible mejorar el rendimiento general del sistema aumentando el número de unidades que realizan la función de activación, lo que conlleva al uso de un FPGA con mucha más capacidad de memoria.

BIBLIOGRAFÍA

ACOSTA BUITRAGO, María Isabel y ZULUAGA MUÑOS, Camilo Alfonso. Tutorial sobre Redes Neuronales Aplicadas a Ingeniería Eléctrica y su Implementación en un Sitio Web. Colombia: Universidad Tecnológica de Pereira, 2000. 360 p.

AMOS, R y OMONDI, Jagath C. Rajapakse, FPGA Implementations of Neural Networks. Holanda: Springer, 2006. 360 p.

ANANTH, Grama; ANSHUL, Gupta; KARYPIS, George y KUMAR, Vipin. Introduction to Parallel Computing. England: Pearson Education, 2003 856 p.

ARMSTRONG, J R y GRAY, F G. Structure Logic Desing with VHDL. London England: Prentice Hall, 1993. 540 p.

ASHENDEN, Peter J. The Designer's Guide to VHDL. 2da ed. Morgan Kaufman, 2002. 458 p.

BASIL, M y SUARDÍAZ MURO, Juan. Nuevas Tendencias En El Diseño Electrónico Digital. Madrid España: Codiseño Hardware/Software, Universidad Alfonso X El Sabio, 2007. 160 p.

BERTONA, Luís Federico. Entrenamiento De Redes Neuronales Basado En Algoritmos Evolutivos. Tesis de grado. 2005. 253 p. (Ingeniero de sistemas) Universidad de Buenos Aires.

BOLETIN MANO AL PECHO. Porque el primer paso para detectar el cáncer de seno está en tus manos, Bogotá, 29 de septiembre de 2008. 3 p.

BROWN, Stephen y VRANESIC, Zvonko. Fundamentals DIGITAL LOGIC WITH VHDL DESIG. Toronto: Mc Graw Hill, 2004. 939 p.

DEL BRÍO, Bonifacio Martín y SANZ MOLINA, Alfredo. Redes Neuronales y Sistemas Borrosos. 3ra ed. México: Alfaomega, 2007. 404 p.

FLOYD Tomas L. Fundamentos de Sistemas Digitales. 7ma ed. Madrid: Prentice Hall, 2002. 1128 p.

FREEMAN James y, SKAPURA, David M. Neural Networks Algorithms, Applications, and Programming Techniques. Wokingham: Addison-Wesley, 1991. 416 p.

GÜICHAL, Guillermo. Diseño Digital Usando Lógica Programable. México: Universidad Tecnológica Nacional Facultad Regional Bahía Blanca, 2005. 172 p.

HERNANDEZ PEREZ, Alberto. Introducción al Lenguaje de Descripción de Hardware VHDL. Cuba: Centro de Investigaciones en Microelectrónica, CUJAE, 2004. 226 p.

HERRERA LOZADA, Juan Carlos. México: Tecnologias Programables, 2007. 41 p.

HOLT, J. L. and HWANG, J. N. Finite-precision error analysis of neural network hardware implementations. Washington, DC, USA: IEEE Transactions on Computers. 1993. 290. p.

LAPLANTE, P. Real-time systems design and analysis. An engineer's handbook. IEEE Press: 1992. 330 p.

MAGDALENO, E.; RODRÍGUEZ, M.; AYALA, A.; MENDOZA, B. y RODRÍGUEZ, S. Metodología para el Aprendizaje de Sistemas Electrónicos Digitales y su Diseño. Bogotá: TAEE, 2004. 410 p.

PEDRONI, Volnei A. Circuit Design with VHDL. London England: MIT Press, 2004. 365 p.

WIDROW Bernard y LEHR, Michael A. Neural networks: Perceptrón, Madaline, and Backpropagation. 2003. 353 p.

WILEY A, John & SONS. Arithmetic and Logic in Computer Systems. Texas: Editorial Wiley Interscience, 2004. 246 p.

NETGRAFÍA

[en línea] Disponible en Internet: http:/gemini.udistrital.edu.co/ comunidad/ profesores/jruiz/jairocd

[en línea] Disponible en Internet: http://eltamiz.com/elcedazo/2008/10/21/ inteligencia-artificial-redes-neuronales

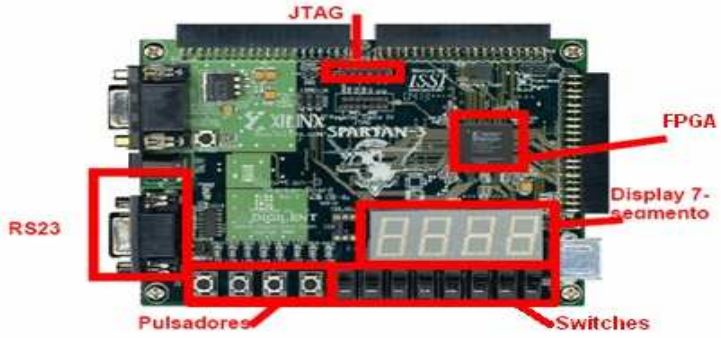
HOWARD DEMUTH, Mark Beale y HAGAN, Martin. Matlab Neural Network Toolbox™ 6 User's Guide, The Math Works Inc. 2009 [en línea] Disponible en Internet: (http://www.mathworks.com). P. 906.

ANEXOS

ANEXO A. PLATAFORMA DE DESARROLLO HARDWARE SPARTAN 3

La plataforma de desarrollo *Spartan-3* de Xilinx trabaja con una frecuencia de reloj de 50 MHz impuesta por un oscilador incluido en la propia tarjeta. Dispone, además, de dos módulos de memoria RAM de 256 KB. También incorpora un conjunto de elementos que facilitan el desarrollo de sistemas y permiten diversas aplicaciones, entre ellos se han empleado los siguientes: un puerto RS232 que permite la comunicación con un PC, cuatro botones pulsadores, cuatro displays 7-segmentos que permiten verificar cualquier operación, ocho switches, un puerto de comunicación serie JTAG para la descarga y chequeo de la aplicación en el FPGA. En la figura 46, se pueden observar dichos componentes.

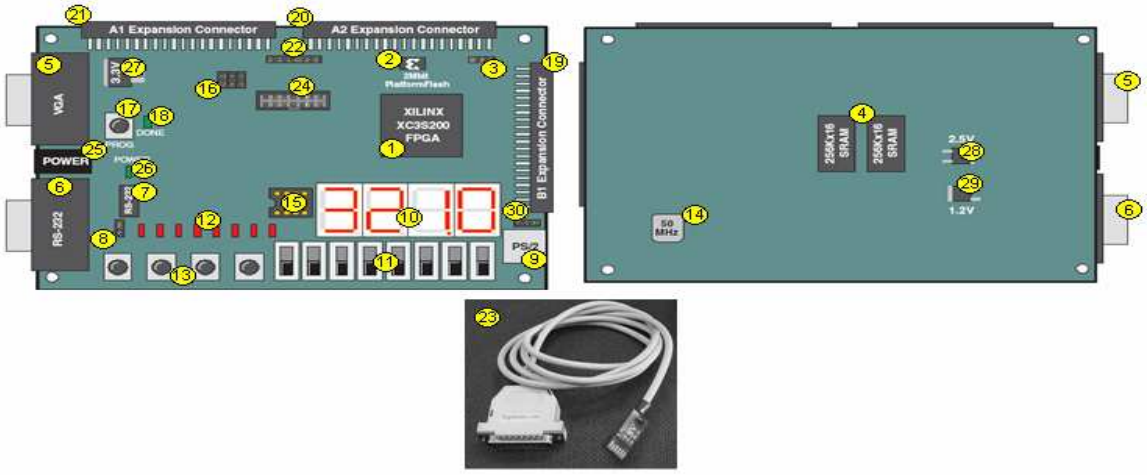
Figura 46. Tarjeta de desarrollo S3.



En la figura 47, se ilustran más detalladamente los componentes que contiene esta tarjeta.

1. Dispositivo FPGA Spartan 3 XC3S200 (XC3S2OOFT256).
2. Memoria flash PROM de 2 Mbits para almacenamiento del programa.
3. Jumper de configuración del FPGA y la PROM.
4. Dos memorias SRAM de 256 por 16 (ISSI IS61LV25616AL-lOT).
5. Conector VGA.
6. Conector RS-232.
7. Corrector de Nivel Max 232.
8. Puerto secundario RS-232.
9. Conector PS2.
10. Cuatro displays de 7 segmentos.
11. Ocho switches
12. Ocho LEDs.
13. Cuatro pulsadores.

Figura 47. Componentes de la tarjeta de desarrollo S3.



Fuente. Spartan-3 starter kit board user guide.

14. Cristal oscilador de 50 Mhz.
15. Base para un cristal oscilador auxiliar.
16. Jumpers para configuración de modo.
17. Pulsador de reconfiguración (carga el programa de la flash al FPGA).
18. LED que indica que el FPGA está configurado correctamente.
19. Conector de expansión 1 de 40 pines (B1)
20. Conector de expansión 2 de 40 pines (A2)
21. Conector de expansión 3 de 40 pines (A1)
22. Conector para el cable de descarga JTAG.
23. Cable para descarga JTAG
24. Conector JTAG (para ser usado con el cable paralelo IV Xilinx, no está incluida en el kit S3)
25. Conector para cable de alimentación.
26. LED de encendido.
27. Regulador de voltaje a 3.3 V.
28. Regulador de voltaje a 2.5 V.
29. Regulador de voltaje a 1.2 V.
30. Selector de alimentación para conector PS2 (3.3 or 5 V).

- **Arquitectura de las FPGA spartan 3 de Xilinx.**

Las FPGA Spartan 3 de Xilinx están conformadas por un conjunto de Bloques Lógicos Configurables (CLBs) rodeados por un perímetro de Bloques Programables de entrada/salida (IOBs). Estos elementos funcionales están interconectados por una jerarquía de canales de conexión (Routing Channels), la que incluye una red de baja capacitancia para la distribución de señales de reloj

de alta frecuencia. Adicionalmente el dispositivo cuenta con bloques de memoria RAM de doble puerto, cuyos anchos de buses son configurables, y con bloques de multiplicadores dedicados de 18 X 18 bits.
Los cinco elementos funcionales programables que la componen son los siguientes:

- *Bloques de entrada/salida (Input/Output Blocks–IOBs)*. Controlan el flujo de datos entre los pines de entrada/salida y la lógica interna del dispositivo. Soportan flujo bidireccional, más operación tri-estado y un conjunto de estándares de voltaje e impedancia controlados digitalmente.

- *Bloques Lógicos configurables (Configurable Logic Blocks – CLBs)*. Contienen Look-Up Tables basadas en tecnología RAM (LUTs) para implementar funciones lógicas y elementos de almacenamiento que pueden ser usados como *flip-flop*s o como *latches*.

- *Bloques de memoria RAM (Block RAM)*. Proveen almacenamiento de datos en bloques con dos puertos independientes cada uno.

- *Bloques de multiplicación.* Aceptan dos números binarios de 18 bit como entrada y entregan uno de 36 bits.

- *Administradores digitales de reloj (Digital Clock Managers–DCMs)*. Estos elementos proveen funciones digitales auto-calibradas, que se encargan de distribuir, desfasar en pocos grados (90, 180, y 270 grados), dividir y multiplicar las señales de reloj de todo el circuito.

Las características para los diferentes dispositivos de la familia spartan 3 son mostradas en la tabla 9.

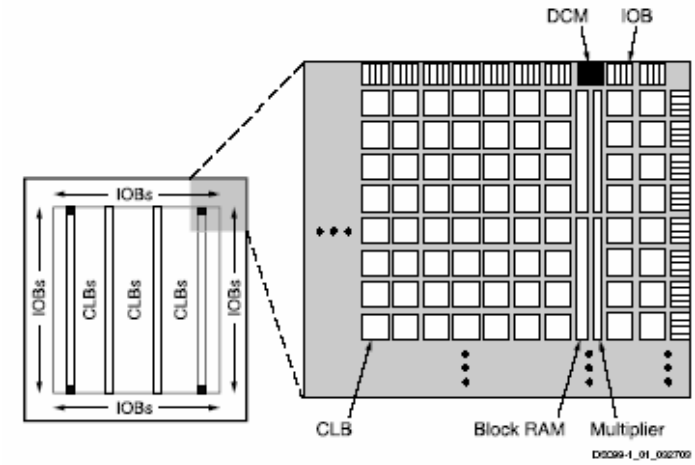
Tabla 9. Características de la familia spartan 3.

Device	System Gates	Equivalent Logic Cells	CLB Array (One CLB = Four Slices)				Distributed RAM Bits	Block RAM Bits	Dedicated Multipliers	DCMs	Maximum User I/O	Maximum Differential I/O Pairs
			Rows	Columns	Total CLBs	Total Slices						
XC3S50	50K	1,728	16	12	192	768	12K	72K	4	2	124	56
XC3S200	200K	4,320	24	20	480	1,920	30K	216K	12	4	173	76
XC3S400	400K	8,064	32	28	896	3,584	56K	288K	16	4	264	116
XC3S1000	1000K	17,280	48	40	1,920	7,680	120K	432K	24	4	391	175
XC3S1500	1500K	29,952	64	52	3,328	13,312	208K	576K	32	4	487	221
XC3S2000	2000K	46,080	80	64	5,120	20,480	320K	720K	40	4	565	270
XC3S4000	4000K	62,208	96	72	6,912	27,648	432K	1,728K	96	4	633	300
XC3S5000	5000K	74,880	104	80	8,320	33,280	520K	1,872K	104	4	633	300

Fuente. Spartan-3 generation configuration user guide.

Los bloques descritos están organizados como se muestra en la Figura 48. Un anillo de IOBs rodea un arreglo regular de CLBs. Atraviesa este arreglo una columna de Bloques de memoria RAM, compuesta por varios bloques, cada uno de los cuales está asociado con un multiplicador dedicado. Los DCMs están colocados en los extremos de dichas columnas.

Figura 48. Arquitectura de la spartan 3.



Fuente. Spartan-3 generation configuration user guide.

ANEXO B. FLUJO DE DISEÑO PARA FPGAs

Durante el proceso de creación de un sistema digital desde el código fuente (esquemáticos, VHDL, etc.) a la implementación en un FPGA hay varios pasos intermedios. Para cada uno de estos pasos se utilizan herramientas de software diferentes que pueden o no estar integradas bajo un ambiente de desarrollo. En muchos casos las herramientas utilizadas en cada paso del diseño son provistas por diferentes empresas. A continuación se describe cada uno de los pasos en el flujo de diseño típico para FPGAs. Dependiendo de las herramientas utilizadas, éste puede tener variaciones o las tareas llamarse con otros nombres.

1. Descripción del diseño. Éste es el paso en el que se describe el diseño usando un lenguaje de descripción de hardware, como VHDL. Muchas herramientas permiten ingresar el diseño, no sólo como HDLs sino también como un diagrama esquemático o estructural, una representación gráfica de una máquina de estados o una tabla de entrada-salida. Estas herramientas simplifican en gran medida el diseño y las tareas del diseñador.

2. Compilado (compile). En este paso, los simuladores actuales compilan el código VHDL a un formato que permite una simulación más rápida y eficaz.

3. Simulación y verificación. Aquí se simula y se evalúa el comportamiento del diseño. La simulación puede hacerse en tres etapas diferentes del diseño. La primera es sobre el código VHDL original para verificar el correcto funcionamiento del diseño. La segunda es después de sintetizar el circuito, simulando la implementación real sobre el FPGA, ya sea con o sin la anotación de tiempos. La tercera etapa, en la cual se puede simular el diseño es después de la Ubicación e Interconexión. Ésta es la más exacta, ya que incluye la información lógica y temporal del diseño.

4. Síntesis (synthesis). En este paso se traduce el VHDL original a su implementación con lógica digital, utilizando los componentes específicos del FPGA que van a utilizarse. Esta traducción puede llegar hasta el nivel más básico de elementos lógicos (CLBs, LUTs, FFs) o hasta un nivel superior, en el cual el diseño se presenta en módulos básicos estándar provistos en una librería por el proveedor de hardware.

5. Ubicación e interconexión (place and route). El FPGA está compuesto por muchos bloques idénticos. En este paso, cada componente del diseño sintetizado se ubica dentro del FPGA. También se interconectan los componentes entre sí y con los pines de entrada-salida.

6. Tareas adicionales. Las tareas adicionales dependen del fabricante y las herramientas. Puede definirse la interconexión de la lógica interna con los pines físicos del FPGA, ingresar condiciones de entorno físico para guiar a la herramienta de Ubicación e Interconexión, seleccionar áreas del FPGA para ubicar los bloques lógicos, etc.

7. Anotación de retardos. Como en todo circuito digital, las señales tienen un retardo de propagación que influye sobre su comportamiento. Con estos retardos puede anotarse el diseño compilado para una simulación que incluye información de temporizado más cercana a la implementación real. Una vez sintetizado el diseño, puede hacerse una estimación de los retardos de propagación que hay entre las señales. Después de la ubicación e interconexión, el cálculo de retardos es mucho más exacto.

8. Generación del archivo de configuración. Después de la Ubicación e Interconexión se genera algún archivo ya sea para poder utilizar el sistema en un diseño más complejo o para programar un FPGA.

9. Configuración del FPGA. Con el archivo binario generado puede configurarse directamente el FPGA a través de alguna de las opciones de configuración. Estas opciones dependen de las herramientas y del dispositivo que se esté utilizando.

10. Programación de memoria (PROM). Muchas FPGA no pueden configurarse de manera permanente y requieren algún tipo de memoria para leer la configuración, lo cual requiere la creación de un archivo específico para grabar el diseño en dicha memoria.

ANEXO C. ISE WEB PACK 7.1

ISE Web Pack 7.1 es una herramienta software gratuita de desarrollo de circuitos digitales, elaborada por la compañía Xilinx, que permite, crear un circuito digital llevándolo a través de cada paso del flujo de diseño para FPGAs.

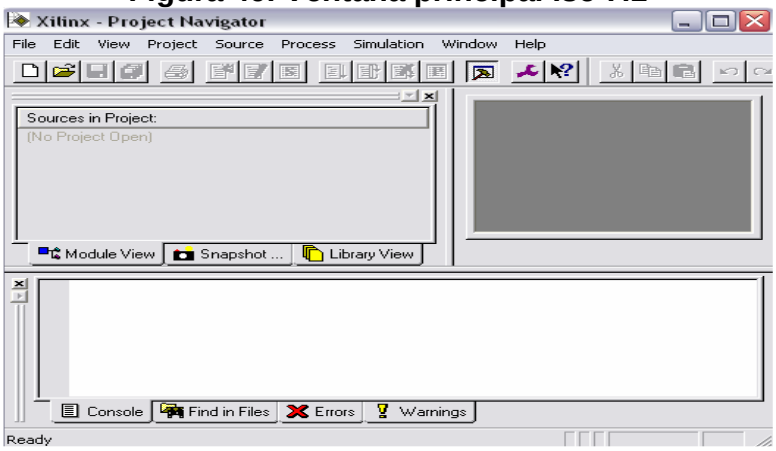
En este software se distinguen 4 ventanas importantes:

- *Sources window*: organizador de proyectos.
- *Process window:* muestra los procesos disponibles para el archivo fuente seleccionado.
- *Transcript window*: muestra mensajes de estado, errores y advertencias.
- *Workplace*: contiene ventanas para múltiples documentos tales como códigos en HDL, reportes, etc.

- **Iniciar y crear un nuevo proyecto en ISE.**

Una vez el programa este correctamente instalado, los primeros pasos a seguir para trabajar con Ise Web Pack 7.1 son los descritos a continuación.
1. El primer paso es arrancar el programa desde inicio/ programas/ Xilinx Ise 7.1/ Project navigator. A continuación se abre una ventana similar a la figura 49.

Figura 49. Ventana principal Ise 7.1

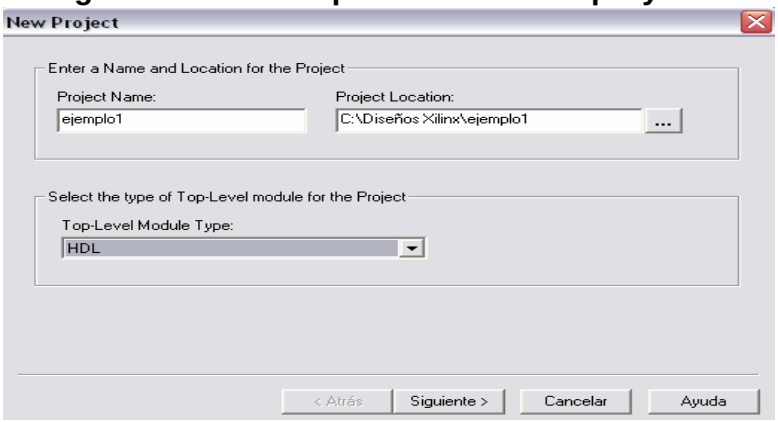


Fuente. Ise 7.1.

2. Crear un nuevo proyecto, para ello Seleccionar *File/ New Project*. Entonces se abre una ventana similar a la mostrada en la figura 50

3. Asignar al proyecto un directorio de ubicación en *Project Location* , un nombre en *Project name* y en *Top-Level Module Type* se debe seleccionar HDL (para trabajar usando lenguajes de descripción de hardware). A continuación hacer clic en <Next> con lo cual aparece una ventana como la mostrada en la figura 51.

Figura 50. Ventana para crear nuevo proyecto.

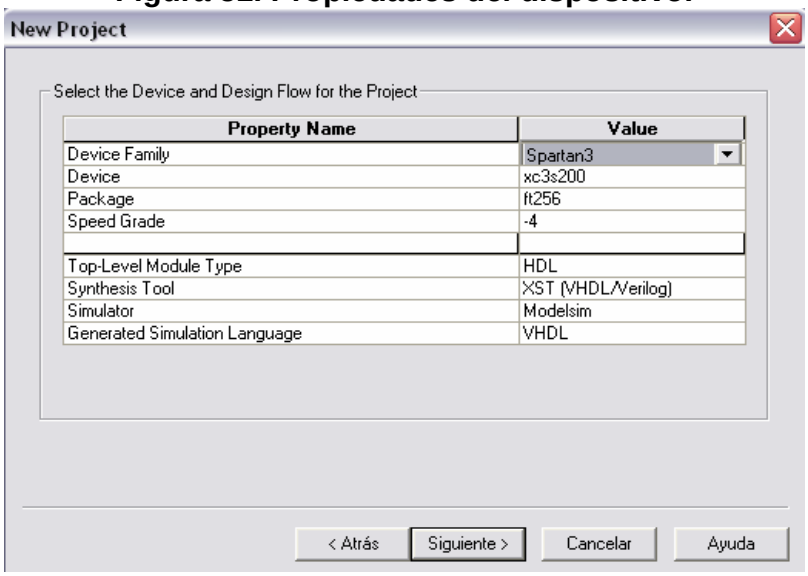


Fuente. Ise 7.1.

4. Escoger las propiedades del dispositivo en el que se va a trabajar y posteriormente hacer clic en <Next>.

5. Obviar los pasos de añadir archivos nuevos, para ello hacer clic en <next> unas pocas veces hasta que aparezca una ventana con el resumen de las propiedades del proyecto. Por último hacer clic en <Finish> para completar la creación del proyecto.

Figura 51. Propiedades del dispositivo.



Fuente. Ise 7.1.

- **Crear un módulo HDL.**

Una vez creado el proyecto, se debe adicionarle a éste, un archivo HDL, esto puede se hecho de dos formas, creando un archivo HDL o adicionando uno creado anteriormente.

A continuación se explica cómo se adhieren módulos existentes y como se crea un nuevo módulo en VHDL.

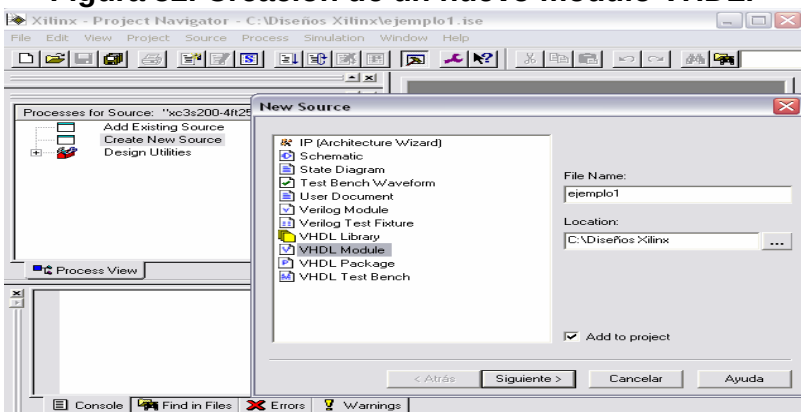
Si se quiere adicionar un módulo ya creado, hacer clic en *Add existing file*, en la ventana de procesos (*Processes window)*, buscar el módulo y hacer clic en <Open>.

Para crear un módulo VHDL se procede de la siguiente forma:

1. Hacer clic en *Create new source* en la ventana de procesos *Processes window.* Esto despliega una ventana como la mostrada en la figura 52.

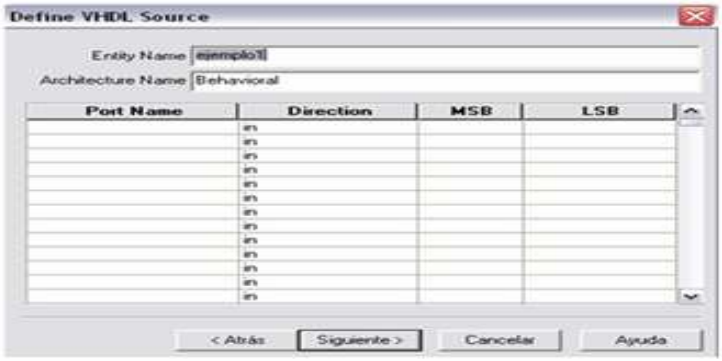
2. Seleccionar *VHDL Module* y luego escribir un nombre para Éste, en *File Name*. No se debe olvidar activar la casilla *Add to Project.* Posteriormente hacer clic en <Next>.

Figura 52. Creación de un nuevo módulo VHDL.



Fuente. Ise 7.1.

Figura 53. Definición de entradas y salidas.



- **Port Name.** El nombre de la entrada/salida.
- **Direction:** Define si el puerto es usado como:
  - ✓ **in:** Pin de entrada.
  - ✓ **out:** Pin de salida.
  - ✓ **in/out:** Pin bidireccional.
- **MSB:** Bit mas significativo.
- **LSB:** Bit menos significativo.

Fuente. Ise 7.1.

3. Por último se definen las entradas y salidas del módulo a desarrollar (también se pueden especificar en el código), para ello, llenar los campos disponibles (mostrados en la figura 53), y luego hacer clic en <Next>.

- **Edición del código.**

Una vez creado el módulo en el proyecto se procede a describir el diseño mediante lenguaje VHDL.

- **Asignación de Pines.**

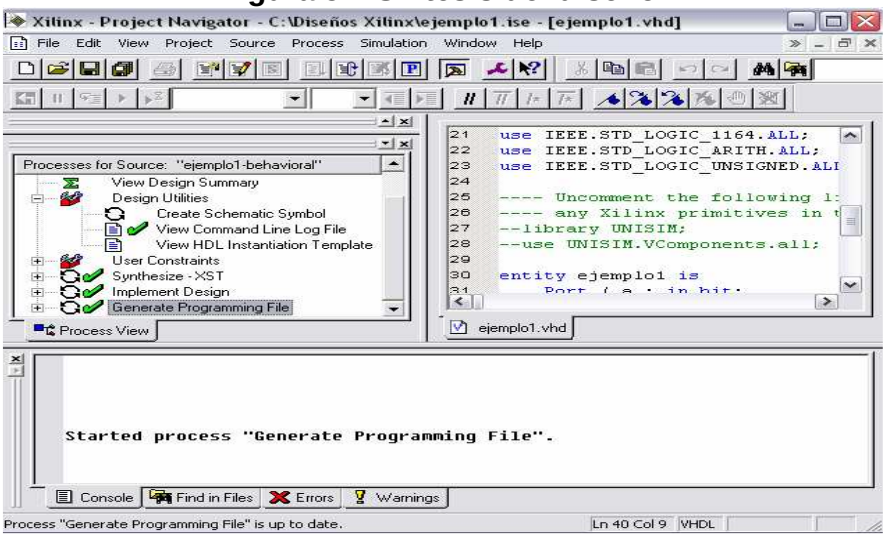
Es necesario asignar físicamente las entradas y salidas del diseño a los pines del dispositivo.

Para hacer esto se accede al menú *User Constraints, Assing Package Pins*, el cual ejecuta la utilería *PACE* que carga la información necesaria para poder signar correctamente los pines del dispositivo en relación al diseño.

- **Síntesis.**

En la lista de procesos disponibles hay una herramienta que permite realizar el proceso de síntesis llamada *Synthesize – XST*. Hacer doble-clic sobre ésta, para ejecutar los procesos necesarios. Este paso permite verificar que la sintaxis del diseño esté correcta y genera el reporte del proceso de síntesis

Figura 54. Síntesis del diseño.



Fuente. Ise 7.1.

Es importante tener en cuenta que al final de todo este proceso las señales correspondientes a *view synthesis report* y *Check syntax* deben aparecer en verde (esto significa que el diseño funciona correctamente) en caso contrario se deberá revisar nuevamente el código del diseño.

- **Simulación.**

La simulación puede ser llevada a cabo de diferentes maneras, una de ellas es crear un módulo especial denominado *Testbench*, la creación y diseño de estos módulos son realizados mediante VHDL y ejecutados en programas especiales para simulación, entre los que se destacan *Modelsim*.

- **Generación de Archivo de Configuración.**

Una vez configurado el proyecto y habiendo especificado todo lo necesario, se procede a generar el archivo de configuración (*.bit) el cual se utiliza para configurar la lógica digital del FPGA. Para esto se puede simplemente dar doble-clic en el menú *Generate Programming File* el cual llama recursivamente los procesos necesarios para generar este archivo.

- **Configuración de FPGA.**

En este punto ya se tiene todo lo requerido para configurar el FPGA, es necesario entonces, utilizar un software que maneje los "Drivers" de los cables que se utilizan para comunicar el PC con la tarjeta de desarrollo. El software utilizado por defecto para configurar los dispositivos Xilinx es Impact y viene integrado dentro del entorno ISE 7.1.

Impact es utilizado para manejar todos los drivers que se instalan con el ISE. Normalmente se utiliza para comunicarse a través del puerto paralelo con el cable *Parallel Cable IV*, así como los cables integrados en los kits de desarrollo de Spartan3.

Para llamar éste programa se necesita desplegar el menú *Generate Programming File* y dar doble-clic en *Configure Device (IMPACT)*. Una vez dentro, lo más conveniente es elegir la opción *Configure devices using Boundary-Scan / Automatically connect to a cable and identify Boundary-Scan chain* ya que esta opción busca los cables y dispositivos conectados y los agrega al proyecto.

Al utilizar la tarjeta Spartan3 Starter kit se despliegan dos dispositivos, uno de los cuales es el FPGA (XC3S200) y la memoria Flash de configuración (XCF02S). Al primero se le asigna el archivo de configuración y al segundo sólo se da clic en el botón <Bypass>.

Para configurar el dispositivo se hace clic-derecho en el mismo y luego se escoge la opción *Configure Device*.