
UNIVERSIDAD AUTONOMA DE QUERETARO
EDUCO EN LA VERDAD Y EN EL HONOR

Universidad Autónoma de Querétaro

Facultad de ingeniería

Maestría en instrumentación y control automático

Diseño e Implementación de una Red Neuronal Artificial en FPGA para la Identificación de Sistemas en línea

Que como parte de los requisitos para obtener el grado de

Maestría en Ciencias en Instrumentación y Control Automático

Presenta:

Ing. Genaro Mendoza Hernández

Dirigido por:

Dr. Luis Alfonso Franco Gasca

Dr. Luis Alfonso Franco Gasca	
Presidente	Firma
M. en C. Alfonso Noriega Ponce	
Secretario	Firma
Dr. Edgar Alejandro Rivas Araiza	
Vocal	Firma
M. en C. José Marcelino Gutiérrez Villalobos	
Suplente	Firma
Dr. Luis Miguel Contreras Medina	
Suplente	Firma
Dr. Aurelio Domínguez González	Dr. Irineo Torres Pacheco
Nombre y Firma	Nombre y Firma
Director de la Facultad	Director de Investigación y Posgrado

Centro Universitario
Querétaro, Qro.
Mayo del 2013
México

Diseño e Implementación de una Red Neuronal Artificial en FPGA para la Identificación de Sistemas en línea

Ing. Genaro Mendoza Hernández

Centro Universitario
Querétaro, Qro.
Mayo del 2013
México

RESUMEN

Las redes neuronales artificiales siempre han demostrado su utilidad en la identificación de sistemas gracias a ventajas como la habilidad de emular relaciones de entrada-salida complejas en tiempo real, la capacidad de aproximar cualquier función si es adecuadamente configurada y recientemente la posibilidad de implementarlas usando FPGA que mejora considerablemente su rendimiento, tomando ventaja de la capacidad de ser implementadas en paralelo. Este trabajo se centra en el diseño en FPGA de una red neuronal artificial para la identificación en línea de un sistema, sin embargo, poco se ha hecho para analizar el desempeño de las red debido a su configuración inicial por lo que se propone un diseño estadístico de experimentos del error de aproximación de dos arquitecturas de red neuronal: el perceptron multicapa y un tipo de red neuronal recurrente ("Locally Recurrent, Globally Feedforward") con realimentación en la función de activación. Analizando la varianza del modelo estadístico resultante, en un esfuerzo por generar una metodología de trabajo dedicado a la búsqueda de la mejor red y de la configuración más adecuada en cada caso de estudio. Los resultados de la simulación y el diseño en VHDL de la mejor configuración de red para los datos obtenidos de un motor de corriente directa son reportados.

Palabras Clave: *Diseño de experimentos, perceptron multicapa, inteligencia artificial, redes recurrentes, ANOVA, ARX, VHDL*.

SUMMARY

Artificial neural networks have always proved to be useful in systems identification thanks to advantages such as the ability to emulate complex input-output relationships in real-time, the ability to approximate any function if properly configured and recently the ability to implement them using FPGA which considerably improves its performance by taking advantage of the ability to be implemented in parallel. This work is focused on an artificial neural network FPGA design for a system online identification, however, little has been done to analyze the performance of the network due to its initial configuration so a DoE of the approximation error of two neural networks architectures: the multilayer perceptron and a type of recurrent neural network ("Recurrent Locally, Globally Feedforward") with feedback on the activation function is proposed. Analyzing the variance of the resulting statistical model, in an effort to generate a methodology dedicated to finding the best network and the most appropriate configuration for each case of study. The results of the simulation and VHDL design of the best network configuration for the data obtained from a direct current motor are reported.

Key words: Design of experiments, multilayer perceptron, artificial intelligence, recurrent networks, ANOVA, ARX, VHDL.

AGRADECIMIENTOS.

A mi **Madre**, que con su amor y su bondad me apoyó incondicionalmente.

A mi **Padre**, que me enseño a ser quien soy, cultivando en mi honestidad y constancia para alcanzar este logro.

A mis **hermanos**, cómplices de tantas experiencias y participes de este trabajo.

A mi **Asesor**, el Dr. Luis Alfonso Franco Gasca por todo sus consejo y apoyo para terminar este trabajo.

A mis **Maestros,** pilares fundamentales de mi formación académica y personal.

A mis **Amigos**, compañeros que la vida me regalo para celebrar cada una de los desafíos de nuestro día a día.

A mi **Compañera,** por su compresión y cariño a lo largo de todo este tiempo.

ÍNDICE DE CONTENIDO

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

I INTRODUCCIÓN

En el campo de la identificación de sistemas, se usan ecuaciones diferenciales y algebraicas para describir el comportamiento del sistema real, creando modelos para representarlo. La creación de modelos requiere una cantidad considerable de tiempo y un conocimiento profundo del sistema, sin embargo, La complejidad y las no linealidades del sistema pueden crear modelos incompletos o que no se asemejan lo suficiente al sistema real (Meireles et al., 2003). Estos problemas pueden llegar a ser tan significativos, que la omisión de estos en los esquemas de control tienen efectos negativos en su desempeño (Alanis et al., 2010).

Dada la capacidad de aproximar cualquier función continua sobre un dominio acotado, las redes neuronales son consideradas como una herramienta útil en la identificación (Giró Et Al., 2007). Una red Neuronal es una interconexión de unidades simples de procesamiento, también llamadas neuronas. Cada unidad recibe información a través de sus entradas, éstas procesan la información recibida y emiten una única salida o respuesta que se transmite a múltiples neuronas posteriores. Cada elemento de procesamiento tiene un número de pesos sinápticos o simplemente pesos. Ajustando los pesos de un elemento se puede cambiar el comportamiento del mismo y, por lo tanto, puede también alterarse el comportamiento de la red para alcanzar la relación de entrada salida deseada (Norgaard et al., 2003). Este último proceso es conocido como entrenamiento de la red.

Algunos puntos difíciles de conocer dentro de la teoría de redes neuronales son en su mayoría aspectos específicos de su configuración y entrenamiento, donde se presentan múltiples posibilidades y se carece de recomendaciones definitivas que permitan seleccionar las más convenientes para cada caso, los cuales se mencionaran a continuación: la arquitectura de la red, en el tipo de red o en algunos casos en lo que se refiere a la cantidad de capas, las unidades por capa y los vínculos entre ellas, las funciones de activación que pueden ser lineales, hiperbólicas, sigmoides o una combinación de ellas, los pesos iniciales más apropiados y las técnicas para el proceso de entrenamiento. A pesar de que estos aspectos son objeto de intenso estudio, por el

momento no puede evitarse una tarea de exploración hasta encontrar la combinación más conveniente para cada caso (Giró et al., 2007).

Las aplicaciones prácticas de las redes neuronales se incrementan con el desarrollo de la electrónica, destacando en estas la capacidad de implementar las redes neuronales con procesamiento en paralelo (Giró et al., 2007; Orlowska, 2011), esto es porque una de las mayores prioridades casi en la totalidad de las áreas de aplicación, es la necesidad de realizar procesos con datos de forma rápida. Las redes neuronales se adaptan bien a su implementación en hardware (Monmasson et al., 2011).

Este trabajo se centra en el diseño de una red neuronal artificial en FPGA para la identificación de sistemas en línea y también en la comparación estadística del error de aproximación para dos redes neuronales (red perceptron multicapa y una clase de red neuronal recurrente) para obtener la red neuronal con mejor desempeño y su configuración óptima.

I.1 DESCRIPCIÓN DEL PROBLEMA

En la actualidad los sistemas de control se han convertido en una parte integral de nuestra vida diaria, estos se encuentran en todo desde simple electrónicos para el hogar hasta aviones y naves espaciales. Los sistemas de control pueden tomar muchas diferentes formas pero lo que todos tienen en común es su principal función de manipular un sistema para que se comporte en una forma deseada.

Cuando se diseña un controlador para un sistema en particular, es necesario que se conozca el sistema o la respuesta del este cuando se manipula de diferentes maneras. Es hasta que conocemos nuestro sistema que podemos proponer como el sistema pude ser controlado para mostrar un comportamiento esperado (Norgaard et al., 2003).

Uno de los métodos más comunes para el diseño de sistemas de control en el ámbito industrial lleva más de 60 años siendo usado por su enfoque sencillo y práctico, es el método de Ziegler y Nichols. Éste se basa en llevar a cabo un experimento con el

sistema a controlar para provocar una respuesta particular, después basado en el conocimiento particular de la respuesta obtenida, simples reglas son seguidas indicándonos como el sistema debe de ser diseñado. Éste y algunos métodos similares se han implementado en dispositivos comerciales para la industria y nombrados como “auto-tune” (Ang et al., 2005).

Dada la enorme cantidad y diversidad de sistemas que deben ser controlados, estos métodos no suelen llenar las expectativas debido a la constante exigencia de mejorar el desempeño de los lazos de control dentro de la industria. Por ejemplo, la apreciación de la divisa japonesa ante el dólar en la década de 1990, forzó a la industria japonesa a una reducción drástica de costos, para mantener así su competitividad. Entre las estrategias implementadas exitosamente para salir delante de este problema, estuvieron la evaluación y adopción de tecnologías innovadoras de control de procesos en sustitución del control convencional utilizado. Consecuentemente, hoy en día la industria japonesa cuenta con la mayor relación de control no convencional instalado en el mundo (Paz, 2006).

Por esto la necesidad de usar métodos avanzados de diseño, donde este tipo de métodos requiere un conocimiento más intensivo del sistema a controlar. Esto es, conocer no solo el comportamiento a un cierto estimulo sino mas bien conocer el modelo del sistema a controlar. Éste puede ser conocido de dos formas: una deductiva, que requiere una gran cantidad de tiempo y en algunos casos el nivel de dificultad para llegar al modelo es muy alto, y la segunda, que es usando una colección de datos durante un experimento para inferir el modelo; esta es conocida como identificación de sistemas, donde un modelo satisfactorio puede ser obtenido en poco tiempo (Norgaard et al., 2003).

Paz (2006) define a la identificación de sistemas como parte integral del esquema de control adaptable. En estos controladores se está validando constantemente el modelo durante su funcionamiento (en línea), y el controlador es rediseñado con respecto al modelo actual.

El problema que se va a abordar en este trabajo es de identificar una planta línea: en este caso tomaremos un motor CD, donde el motor va a estar expuesto a cambios en su entrada. El objetivo principal de este trabajo será la identificación de nuestro caso de estudio, pero también observar los efectos de las perturbaciones en el mismo.

I.2 JUSTIFICACIÓN

En la actualidad, el control de procesos representa una industria multimillonaria, solo en el año 2003 se vendieron 9,200 mdd en sistemas de control distribuido alrededor del mundo, cifra que se espera que siga creciendo (Paz, 2006).

El vínculo entre el control de procesos y la industria ha sido de gran importancia para el desarrollo y calidad de los procesos industriales. Como el análisis que hace Thomas (2004) a la industria química mostrando los múltiples beneficios (económicos, ambientales, de ahorro de energía, de seguridad, de flexibilidad en los procesos, de vida en los equipos) que el control le ha dado.

El control proporcional-integral-derivativo (PID) lleva muchos años con un gran éxito dentro de la industria, con más del 90% de controles PID o PI implementados alrededor del mundo, las características más importantes con que este controlador cuenta son: lo económico de implementar, por la gran cantidad de productos comerciales que se tienen actualmente, y su alta confiabilidad (Espinosa et al., 2006; O´Dwyer, 2006).

A pesar de que el PID tiene tiempo siendo un referente en la industria, aun se tiene mucho por mejorar ya que se estima que más del 30% del total de lazos implementados opera en modo manual, mientras que otro tanto similar opera con las ganancias que por defecto configura el fabricante del controlador (Nallasivam et al., 2008; O´Dwyer, 2006), hay muchas causas de estos problemas, la principales son: una mala sintonización del controlador por la incapacidad del operador, cambio con el tiempo del proceso (cuando no se actualizan dentro de estos esquemas) y no

linealidades en el proceso (procesos que le resultan complejos a los esquemas clásicos).

Los cambios con el tiempo y las no linealidades afectan el proceso, y aunque no siempre son significativos, en otros sistemas y procesos los cambios pueden ser bastante importantes. Haciendo uso de controladores con parámetros fijos, hablando específicamente los PID, sea inaceptable o en algunos casos imposible.

Es en estas áreas de oportunidad donde múltiples publicaciones alrededor del mundo buscan la forma de mejorar el desempeño del controlador, ya que mejorar el PID es una mejora directa en múltiples campos donde es utilizado. Una de las soluciones más directas en la mejora del PID, es encontrar las constantes de sintonización apropiadas a cada aplicación (Espinosa et al., 2006).

Muestra de esto es la cantidad de reglas hoy disponibles para poder sintonizar un PID, por ejemplo en una sola fuente se tiene una recopilación de más de 250 reglas de sintonización (O´Dwyer, 2006). Pero nos damos cuenta del problema dentro de la sintonización del PID es que son demasiadas reglas, siendo que en realidad no existe un método de sintonización que sea el adecuado para todas las aplicaciones, este método dependerá completamente del proceso a controlar (Espinosa et al., 2006).

Es por esto que el control adaptable tiene una gran ventaja, ya que por si mismo se adecua a los cambios del sistema, una de las vertientes más importantes de los controles adaptables tiene que ver precisamente con la búsqueda de los mejores parámetros para reconfigurarse. En el caso de un control adaptable que toma un PID como base para su funcionamiento, esta búsqueda se resumiría en encontrar las ganancias (Proporcional, Integral y diferencial) que cumplan con el mejor desempeño.

Este trabajo se implementará en un FPGA por sus numerosas ventajas dentro del campo de los sistemas de control como son: flexibilidad en el diseño y una mejora considerable en el rendimiento de los algoritmos implementados bajo esta tecnología, esto por la capacidad de ser implementados en paralelo. En (Monmasson y Cirstea, 2007) tenemos una gran cantidad de aplicación de FPGA en el campo de control industrial, en este articulo se resalta la necesidad y el interés de la industria de mejorar

el funcionamiento de los equipos de control, así como la reducción de tamaño y consumo de energía dentro de los procesos donde estos son un factor importante, como los sistemas embebidos. Dentro de (Monmasson y Cirstea, 2007) se concluye que la implementación de tecnología de control en FPGA puede responder a los actuales y futuros retos dentro de la industria (Monmasson et al., 2007, 2011).

I.3 HIPÓTESIS Y OBJETIVOS

I.3.1 Hipótesis general

Es posible diseñar e implementar una red neuronal artificial en FPGA utilizando arquitecturas paralelas, optimizando los recursos y el número de cálculos para lograr la identificación en tiempo real de un sistema no lineal, logrando reducir el tiempo de procesamiento de datos cuando se le compare con un sistema similar basado en software.

I.3.2 Objetivo general

El objetivo de esta tesis es diseñar una red neuronal artificial para la identificación en línea de sistemas que pueda ser implementada en una sola FPGA.

I.3.3 Objetivos específicos

1. Diseñar una red neuronal artificial en FPGA capaz de identificar en línea la planta propuesta.
2. Programar y depurar la arquitectura de la red neuronal en software, donde ésta debe de ser eficiente en la identificación con el mínimo tamaño posible en la arquitectura, y un buen manejo de los cálculos para que pueda ser implementado.

3. Validar la implementación de la red neuronal y comparar los resultados obtenidos con los obtenidos en la simulación en la PC.

4. Proponer una metodología basada en el diseño de experimentos para la selección de la mejor configuración para los factores seleccionados.

II ANTECEDENTES

II.1 REDES NEURONALES ARTIFICIALES

El campo de las redes neuronales artificiales (RNA) tiene sus raíces en la neurobiología y en el interés que los científicos han mostrado en intentar emular el funcionamiento del cerebro, persiguiendo la construcción de algoritmos capaces de procesar información de una manera similar a éste. Sin embargo, dada la enorme complejidad del funcionamiento del cerebro humano, se reduce a las RNA a una modesta aproximación de este funcionamiento. Donde, sus principales características están inspiradas en el conocimiento científico existente sobre la estructura y forma de funcionamiento de las neuronas biológicas.

Una RNA es una interconexión de unidades simples de procesamiento, también llamadas neuronas. Cada unidad recibe información a través de sus entradas, éstas procesan la información recibida y emiten una única salida o respuesta que se transmite a múltiples neuronas posteriores. Cada elemento de procesamiento tiene un número de pesos sinápticos o simplemente pesos. Ajustando los pesos de un elemento se puede cambiar el comportamiento del mismo y, por lo tanto, puede también alterar el comportamiento de total de la red para alcanzar la relación de entrada salida deseada (Norgaard et al., 2003). Este último proceso es conocido como entrenamiento de la red.

Los primeros estudios en el campo de las redes de neuronas fueron realizados por McCulloch y Pitts (1943), estudios que mostraron la habilidad de un grupo de neuronas conectadas para llevar a cabo la implementación de ciertas funciones lógicas, modelando también una RNA simple mediante circuitos eléctricos.

Años más tarde, Rosenblatt (1958) desarrollo el Perceptrón, que fue la primera arquitectura de RNA con capacidad de aprendizaje. El mismo autor mostró que, modificando los valores asociados a las conexiones, la red era capaz de aprender a responder según unas pocas funciones lógicas.

Posteriormente surgieron críticas (Minsky y Papert, 1969) que frenaron, hasta la primera parte de la década de los 80, el desarrollo en la investigación de RNA. Estos autores probaron formalmente (matemáticamente) como el Perceptrón no era capaz de resolver problemas relativamente sencillos, como por ejemplo el aprendizaje de una función no lineal. Sin embargo, las RNA volvieron a renacer principalmente debido a Hopfield (1982), el cual demostró que redes de neuronas totalmente conectadas tenían una gran capacidad para procesar información.

Años más tarde, exactamente en 1988, Kolgomorov demostró que toda red que disponga por lo menos una capa oculta y contenga en ella una adecuada cantidad de neuronas convenientemente entrenadas, tendrá el carácter de aproximador universal, esto es que será capaz de reproducir cualquier función continua y no lineal que pueda estar definida en cierto espacio (Giró et al., 2007).

Uno de los aportes más importantes en identificación mediante redes neuronales fue publicado en la década de los 90, cuando Narendra y Parthasarathy (1990) demostraron que las redes neuronales identifican sistemas no lineales, y además son útiles para su control. En éste trabajo se concluyó su capacidad de emular sistemas no lineales, aun sin conocimiento previo de ellos. Tal ventaja reduce en gran medida la cantidad de trabajo dedicado en la primer parte del diseño de casi cualquier controlador; la identificación del sistema (Norgaard et al., 2003).

Otra característica importante de las redes neuronales es que pueden adaptarse a los cambios en los parámetros de la planta que emulan (Bose, 2007). Esto es por el aprendizaje en tiempo real con que algunas redes neuronales trabajan. Esta capacidad destaca aun con datos imprecisos o incompletos, debido a que la información está distribuida en las conexiones entre las neuronas y a que en este tipo de almacenamiento existe cierto grado de redundancia. Se desarrolla más el tema de la redundancia en las RNA en (Chandra y Singh, 2003), donde, se tiene un estudio que demuestra que la redundancia es un factor condicionante tanto, a la tolerancia a ruido como a la capacidad de generalización de las redes neuronales.

En la actualidad existe un gran número de RNA, siendo las que más se destacan: el perceptron (Rosenblatt, 1958), la Adaline (Widrow y Hoff, 1960), Las redes de Hopfield (Hopfield, 1982), el perceptron multicapa (Rumelhart et al., 1986), la red de Jordan (Jordan, 1986), las redes de base radial (Moody y Darken, 1988), la red de Elman (Elman, 1988) y los mapas de Kohonen (Kohonen, 1990), etc.. Ésta diversidad radica principalmente en las múltiples aplicaciones donde las RNA se usan, como son: clasificación (Martins et al., 2007), reconocimiento de patrones (Polat y Yildirim, 2010), filtrado (Kabir et al., 2010), predicción (Karali et al., 2007), control (Giró et al., 2006) e identificación (Haber y Unbehauen, 1990; Alataris et al., 2000; Mastorocostas y Theocharis, 2002).

Extensos estudios de las capacidades de modelado y predicción de las RNA han sido reportados, incluso comparando el desempeño de varias de estas para la identificación come en (Carling et al., 1994), quedando claro las diferentes capacidades de las mismas.

De la cantidad de RNA mencionadas, actualmente, las más utilizadas en aplicaciones de modelado de sistemas no lineales son las llamadas redes estáticas, las cuales incluyen el perceptron multicapa y las redes de base radial (Norgaard et al., 2003). La popularidad de dichas arquitecturas no sólo es debido al hecho de que ambas poseen la propiedad de ser aproximadores universales, propiedad imprescindible cuando se trata el problema de la modelización y el control (Giró et al., 2007), sino que también es debido a la existencia de algoritmos de aprendizaje específicos para estas arquitecturas que permiten un entrenamiento sencillo y con éxito en la mayor parte de los casos. Es importante hacer notar que la mayor parte de estas aplicaciones capturan la dinámica del proceso real mediante el uso de líneas de retardos en las entradas y salidas del modelo (Narendra y Parthasarathy, 1990).

La identificación de modelos no lineales utilizando las RNA llamadas recurrentes ha sido también estudiada por diferentes autores, entre ellos (Prokhorov, 2007; Gonzalez y Tang, 2010; Yuan et al., 2010), debido a la presencia de conexiones retroalimentadas en las redes recurrentes, no es necesario representar la información temporal utilizando las entradas y salidas anteriores del proceso para identificar, a diferencia de los

modelos de identificación con RNA estáticas. En estas RNA basta utilizar como entradas a la red los valores actuales de las variables de entrada y salida del proceso (Norgaard et al., 2003).

En (Prokhorov, 2007) se destaca el alcance que las RNA recurrentes tienen para la identificación de sistemas, el autor usa un método de entrenamiento llamado SMD (Stochastic meta-descent) por sus siglas en inglés. Con éste, se toma ventaja de su mejor velocidad en la fase de aprendizaje, haciéndolo una alternativa al algoritmo de retropropagación del error.

Una característica que le da mucho impulso a las redes neuronales en nuestros días, es la capacidad de implementarlas con procesamiento en paralelo realizado por las neuronas artificiales (Giró et al., 2007), esto es porque una de las mayores prioridades casi en la totalidad de las áreas de aplicación, es la necesidad de realizar procesos con datos de forma muy rápida. Las redes neuronales se adaptan bien a esto debido a su implementación en hardware (Monmasson et al., 2011).

Generalmente las redes neuronales son implementadas usando software, pero dentro de este, no se puede llegar a tener las mismas prestaciones de procesamiento que un FPGA nos brinda. Normalmente la velocidad de las implementaciones basadas en software es baja y depende del equipo de cómputo. Además, una ventaja adicional del FPGA es para aplicaciones donde una alta portabilidad es necesaria y la implementación de un PC no es adecuada (Monmasson et al., 2007, 2011). Como prueba, hay muchas diferentes implementaciones de redes neuronales en FPGA (Lyhne et al., 2008; Lin y Hung, 2009; Polat y Yildirim, 2010) que resaltan estas ventajas. En (Espinosa et al., 2006), se presenta la implementación de la red neuronal usando la lógica binaria estándar en FPGA. En (Giró et al., 2006), se discuten las técnicas para la implementación de una MLP (Perceptron Multicapa) en un FPGA y el impacto de los diferentes formatos de aritmética. En (Lin y Hung, 2009), se describe la implementación de una red neuronal Wavelet en FPGA.

La implementación de redes neuronales en FPGA está creciendo considerablemente por sus numerosas ventajas como son: flexibilidad en el diseño y una mejora

considerable en el rendimiento de las RNA implementadas bajo esta tecnología, esto, por la capacidad de ser implementadas en paralelo (Monmasson y Cirstea, 2007).

Es en la implementación en la FPGA donde múltiples decisiones deben de ser tomadas, como es el caso de (Omondi, 2006) donde se habla del mayor problema encontrado al implementar un red neuronal artificial: la precisión que los datos van a tomar contra la cantidad de recursos que se van a consumir dentro del FPGA. Resumiendo, esto es como balancear entre la necesidad de precisión numérica, que es importante para la fidelidad de la red y la velocidad de de convergencia, y el costo de un mayor número de celdas lógicas.

Algunos puntos difíciles de conocer dentro de la teoría de redes neuronales son en su mayoría aspectos específicos de su configuración y entrenamiento, donde se presentan múltiples posibilidades y se carece de recomendaciones definitivas que permitan seleccionar las más convenientes para cada caso, los cuales se mencionaran a continuación:

- La arquitectura de la red, en el tipo de red o en algunos casos en lo que se refiere a la cantidad de capas, las unidades por capa y los vínculos entre ellas.
- Las funciones de activación que pueden ser lineales, hiperbólicas, sigmoides o una combinación de ellas.
- Los pesos iniciales más apropiados.
- Las técnicas para el proceso de entrenamiento.

A pesar de que estos aspectos son objeto de intenso estudio, por el momento no puede evitarse una tarea de exploración hasta encontrar la combinación más conveniente para cada caso (Giró et al., 2007).

III MARCO TEÓRICO

III.1 LA NEURONA

La neurona, es el elemento fundamental de funcionamiento de las RNA. Hay tres elementos básicos que componen el modelo de una neurona: los pesos sinápticos, la función de entrada y la función de activación.

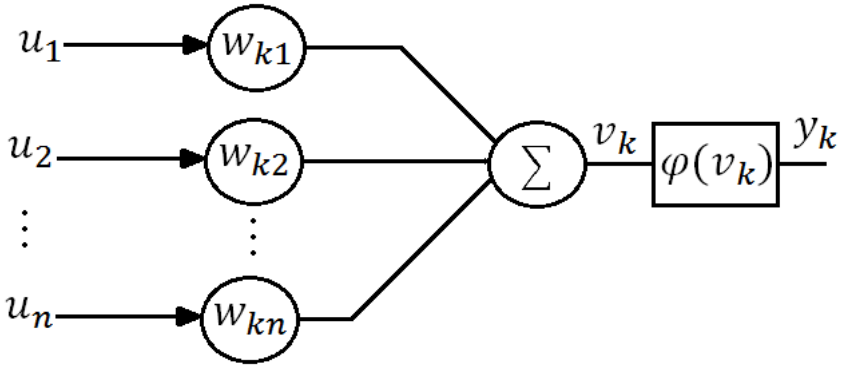


Figura III.1 Modelo de una neurona.

En términos matemáticos, una neurona puede ser definida por el siguiente par de ecuaciones:

III.1

III.2

Cada conexión es caracterizada por un peso sináptico, específicamente la señal de entrada es multiplicada por el peso . Por consiguiente los pesos, que generalmente no están restringidos, cambian la medida de influencia que tienen los valores de entrada en la respuesta de la red.

La sumatoria, suma cada una de estas multiplicaciones para obtener , para después evaluar éste dentro de la función de activación , dando como resultado la salida del sistema .

A continuación se da una descripción más amplia de cada uno de los elementos que componen una neurona.

(i) Vector de entrada

Es denotado por , donde es el número de entradas a la RNA y a su vez la dimensión de , este vector son los datos con los que va operar la neurona, estas entradas pueden ser dadas por el medio ambiente o ser salidas de neuronas anteriores. Cabe hacer notar que si estamos hablando de identificación de funciones o control de sistemas , donde ; es la variable del tiempo, según sea el caso, continuo o discreto, para esta situación las entradas generan una matriz donde n es la dimensión del vector de entradas y m es la dimensión del subespacio , a esta m generalmente se le conoce como número de iteraciones.

(ii) Pesos sinápticos

Al ser capturados los datos de entrada estos son propagados a través de la red, en el proceso de propagación cada componente del vector de entrada es multiplicada por una variable , la cual aumenta o atenúa la señal de entrada, a se le conoce como peso sináptico o simplemente peso, estos pesos no tienen el mismo valor siempre sino que se van modificando según se requiera para tener un mejor desempeño, posteriormente puede haber una convergencia y entonces estar fijos. Cuando se habla de que una red es capaz de aprender se refiere al hecho de poder modificar sus pesos , el conjunto de pesos genera una matriz , es decir, es la kj-ésima componente de la matriz de pesos .

(iii) Operador de suma

Realiza la adición de los productos , la operación aquí descrita constituye una combinación lineal generando un campo local inducido (CLI) , es decir:

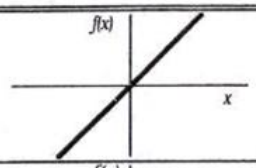
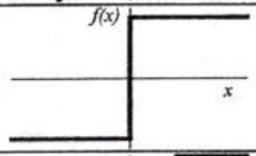
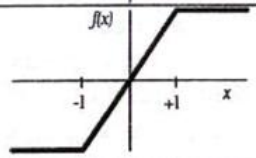
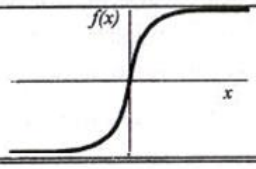
III.3

(iv) La función de activación

Denotada por , es definida como una función para limitar la amplitud de la salida de la neurona. Sin embargo, en el sentido en el que estamos tratando las RNA, la razón de ser de la función de activación es la de proporcionar un comportamiento no lineal a la neurona y así poder aproximar un mayor número de funciones.

La funciones Identidad, Escalón, Lineal a Tramos, Sigmoidea o Tangente Hiperbólica son utilizadas como funciones de activación. Las tres últimas son aptas para la aproximación de funciones, mientras, el escalón tiene una gran aplicación en reconocimiento de patrones (Norgaard et al., 2003). En la mayoría de los casos se ha encontrado que la forma exacta de la función tiene poco que ver con la capacidad de aproximar de la red, aunque sí tiene que ver con la velocidad de convergencia en el entrenamiento.

Tabla III.1Funciones de activación.

	Función	Rango	Gráfica
Identidad	$y = x$	$[-\infty, +\infty]$	
Escalón	$y = sign(x)$ $y = H(x)$	$\{-1, +1\}$ $\{0, +1\}$	
Lineal a tramos	$y = \begin{cases} -1, & si\ x < -l \\ x, & si\ +l \le x \le -l \\ +1, & si\ x > +l \end{cases}$	$[-1, +1]$	
Sigmoidea	$y = \frac{1}{1+e^{-x}}$ $y = tgh(x)$	$[0, +1]$ $[-1, +1]$	

Dentro de las aplicaciones de las RNA como aproximador de funciones y control de procesos, se destaca el uso de las funciones de activación Sigmoidea y Tangente Hiperbólica por sus características: la primera derivada siempre positiva, y la segunda que toma como entrada cualquier valor entre menos infinito y más infinito y genera una salida entre 0 y 1. A este tipo de funciones se les llama a veces “función de compresión” (Cowan, 1967).

III.2 ARQUITECTURA DE LAS REDES NEURONALES ARTIFICIALES

La arquitectura o estructura de las RNA consiste en la organización y disposición de las neuronas en red formando capas o agrupaciones de neuronas. En este sentido, los parámetros fundamentales son: el número de capas, el número de neuronas por capa y el tipo de conexiones entre neuronas. Donde el tipo de conexión de las redes puede clasificarse como “Feedforward” o recurrentes.

En las redes “Feedforward”, todas las señales neuronales se propagan hacia delante a través de las capas de la red. No existen conexiones hacia atrás ni tampoco

recurrentes. En estas redes ninguna salida de las neuronas es entrada de neuronas del mismo nivel o de niveles precedentes (Hertz, 1991). En estas redes, el flujo de información es generalmente síncrono, es decir todos los estados de las neuronas se actualizan al mismo tiempo, o siguiendo alguna secuencia determinista. Es por esto que las redes “Feedforward” también son llamadas redes estáticas (Pham y Xing, 1995).

Como se detallara más adelante, cuando las salidas pueden ser conectadas como entradas de neuronas de niveles previos o del mismo nivel, incluyéndose ellas mismas, la red es llamada recurrente o dinámica.

En los años recientes, se cuenta con un número importante de arquitecturas de RNA. A continuación, se detallan cuatro arquitecturas (o estructuras): redes de una sola capa, redes multicapa, redes recurrentes y redes de rejilla.

III.2.1 Redes estáticas

(i) Redes de una sola capa

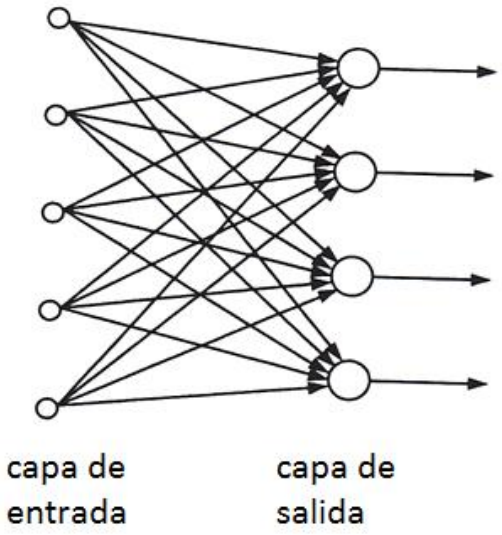


Figura III.2 Arquitectura de redes de una sola capa.

La forma más simple de una RNA es la que tiene una sola capa de neuronas que procesa la información, estas tienen una capa de entrada que se conectan solamente con la capa de salida (procesamiento), pero no de forma contraria. Esta es llamada red de una sola capa, aunque contiene en realidad dos capas, pero solo en una de ellas se procesa información.

(ii) Redes multicapa

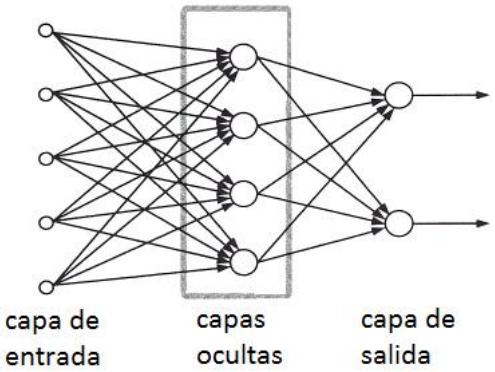


Figura III.3 Arquitecturas de redes multicapa.

Las redes multicapa tienen una capa de entrada, una o muchas capas ocultas y una capa de salida. Cada una de estas capas consiste en neuronas, donde cada una de las neuronas está conectada a neuronas en capas posteriores, es decir, son del tipo “Feedforward”. En éstas, el número de neuronas en la entrada, define la dimensión del espacio de que se asocia a la entrada, y el número de neuronas en la capa de salida, dimensiona el espacio de salida en el que la red mapea la entrada (Zurada, 1992).

En una red neuronal “Feedforward”, la arquitectura total es alcanzada por conexiones intermedias de una capa a otra. Donde, las conexiones intermedias dependen de dos factores, el primero, es el peso de la conexión que transforma la salida de las neuronas en capas anteriores en una entrada para la neurona y la segunda, es la función de activación de la neurona. En estas redes, las neuronas tienen una fuerte restricción en la conectividad, que consiste en no permitir conexiones entre neuronas creando ciclos o bucles. Esta restricción no limita la capacidad de la red

en tareas donde los patrones o entradas a procesar sean estáticos, es decir, patrones en los que no interviene la variable de tiempo. Aun así, es posible, por medio de una serie de estrategias, utilizar dichas estructuras estáticas para procesar sistemas dinámicos (Narendra y Parthasarathy, 1990).

La estrategia para que las redes “Feedforward” procesen sistemas dinámicos es muy simple, y consiste en que las entradas a la red estén compuestas, no sólo del valor del patrón en un determinado instante de tiempo, sino también de una cierta historia de dicho patrón, es decir tomar este patrón en instantes de tiempo pasados. De esta forma, la información temporal puede ser procesada por la red, aunque no se puede olvidar que dicho procesamiento es debido a los patrones de entrada, y no a la propia arquitectura de la red. Esta estrategia ha sido ampliamente utilizada en la modelización y el control de procesos dinámicos, (Narendra y Parthasarathy, 1990), (Chen y Billings, 1992), entre otros.

III.2.2 Redes dinámicas

Una red recurrente tiene por lo menos un lazo de retroalimentación que la distingue de las redes “Feedforward”. Las redes recurrentes pueden ser redes de una sola capa o redes multicapa, donde, cada neurona puede retroalimentar su señal de salida de regreso a la entrada de todas las demás neuronas. Una clase de redes recurrentes con neuronas ocultas es mostrada en la figura 2.4.

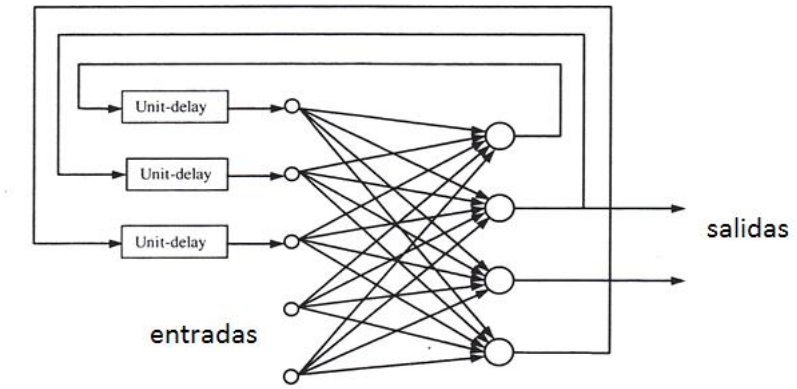


Figura III.4 Arquitectura de una red recurrente.

La presencia de estos lazos de retroalimentación tiene un profundo impacto en la capacidad de aprendizaje de las RNA, y en su rendimiento general. Por lo que hace a las redes recurrentes una alternativa para procesar patrones o entradas dinámicas.

El flujo de información en este tipo de arquitecturas puede ser asíncrono, es decir que los estados de cada neurona se actualizan de acuerdo al comportamiento interno de cada neurona. Así, la secuencia de actualización de pesos resulta estocástica (Warwick et al., 1992).

Estas redes se utilizan con la finalidad de introducir elementos dinámicos y temporales en las RNA. Las redes recurrentes son redes donde sus conexiones no están, en principio, sometidas a ninguna restricción. Esto permite, tanto la existencia de conexiones de una neurona a ella misma, como entre neuronas de una misma capa o entre neuronas de capas diferentes. Las arquitecturas de redes recurrentes incorporan, por tanto, lazos en las conexiones entre las neuronas, lo cual permite que la salida de una neurona de la red no sólo dependa del patrón de entrada, sino también de los estados anteriores de todas las neuronas conectadas a ella, provocando así un comportamiento dinámico en la red y facilitando el procesamiento de la información. Estas redes fueron introducidas y discutidas en los trabajos de Hopfield (1982), surgiendo así la idea general de cómo trabajan las RNA recurrentes.

Las RNA recurrentes se pueden dividir en tres grupos, redes parcialmente recurrentes, redes totalmente recurrentes y “localmente recurrentes, globalmente Feedforward” (LRGF).

En las redes parcialmente recurrentes, se tiene la mayoría de las conexiones no recurrentes, aunque existe un número pequeño de conexiones recurrentes cuidadosamente seleccionadas, que permiten a la red “recordar” el nivel de activación en un pasado de un cierto grupo de neuronas. Como ventaja de éstas es que la presencia de recurrencias no complica en demasía el entrenamiento de la red, pues en la mayoría de los casos los pesos asociados a dichas conexiones son fijos. En estas redes aparecen una serie de neuronas especiales, llamadas neuronas de contexto, que son las receptoras de las conexiones recurrentes. Es decir, funcionan como una

memoria donde se almacena el estado de una determinada capa o conjunto de neuronas, obtenido en instantes de tiempo anteriores. Las primeras se caracterizan porque las neuronas de contexto reciben una copia de la capa de salida y otra de sí mismas. En la red de Elman, dichas neuronas memorizan la activación de las neuronas de la capa oculta.

Las arquitecturas completamente recurrentes corresponden a estructuras neuronales en las cual se permite que cualquier neurona de la arquitectura presente conexiones hacia cualquier otra neurona (Olurotimi, 1994; Lin y Lee, 1996). En estas arquitecturas las conexiones recurrentes no se consideran fijas, sino que existen pesos asociados a ellas, produciéndose así un aumento considerable del número de parámetros ajustables de la red de neuronas. Esto significa aumentar la capacidad de representación de información de la red, aunque por otra parte complica su aprendizaje.

Por último, Tsoi y Back (1994) describen al hibrido de arquitecturas como arquitecturas “localmente recurrentes, globalmente “Feedforward”. Corresponden a arquitecturas que se basan en el esquema de red “Feedforward”, pero construidas mediante neuronas que presentan realimentaciones a nivel local. Dado que la estructura global corresponde a una red tipo “Feedforward”, la clasificación en esta arquitectura se basa en la topología de las realimentaciones a nivel de neuronas.

(i) Redes de rejilla

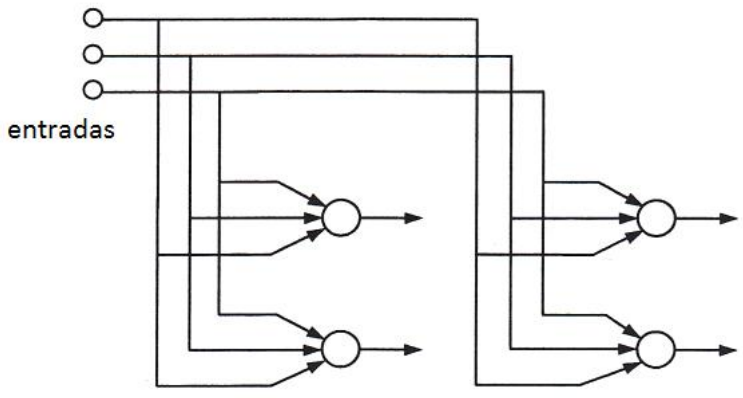


Figura III.5 Arquitectura de la red de rejilla.

Una red de rejilla puede consistir en arreglo dimensional de neuronas de una, dos o múltiples dimensiones. La dimensión en las redes de rejilla se refiere al número de espacios en el que la gráfica reside. Un grupo de nodos fuente en la red, proveen las señales de entrada al arreglo. La Figura III.5 muestra un arreglo de rejilla, de dos dimensiones con dos neuronas con tres nodos fuentes para cada neurona. Una red de rejilla es realmente una red “Feedforward”, con las neuronas de salida acomodadas en filas y columnas.

III.3 APRENDIZAJE Y APROXIMACIÓN

El aprendizaje dentro de la teoría de las RNA se define como el proceso mediante el cual la red modifica su respuesta, o respuestas, ante determinadas entradas. Éste proceso conocido también como fase de entrenamiento, consiste en la modificación paulatina de los parámetros ajustables de la red (pesos de conexión), modificación que se realiza en base a un cierto criterio establecido y que permite que la red “aprenda” a dar las respuestas adecuadas en función de los patrones de entrada presentados (Norgaard et al., 2003).

Generalmente, para la identificación de sistemas con RNA, el criterio de adaptación de la red consiste en minimizar las diferencias entre las respuestas de la red, para cada uno de los patrones de entrada, y las respuestas deseadas para estos patrones. Estas respuestas deseadas actúan como guía para que la red aprenda a realizar una determinada tarea. La determinación de los parámetros ajustables de la red puede tratarse como un problema de optimización para minimizar la siguiente función error:

— III.4

Siendo el número de patrones disponibles, y el error asociado a cada patrón .

Suponiendo que la red tiene m neuronas de salida y que para el patrón k-ésimo genera las salidas , siendo las salidas deseadas correspondientes, el error asociado a dicho patrón se define como:

– III.5

Dado que las salidas de la red dependen del conjunto de parámetros ajustables W, el error es también función de dichos parámetros, pudiendo escribir .

La presencia de no lineales en la RNA hace que su respuesta sea no lineal con respecto a los parámetros ajustables, por lo que el problema es determinar el conjunto de parámetros que minimizan la función error , es un problema de minimización no lineal y, como consecuencia, tienen que utilizarse técnicas de optimización no lineales. Dichas técnicas están generalmente basadas en una adaptación de los parámetros siguiendo una cierta dirección de búsqueda. En el contexto de redes de neuronas, la dirección de búsqueda más comúnmente usada es la dirección negativa del gradiente de la función , (—) pues esta es la dirección en la que la función decrece.

Basándonos en el método de descenso del gradiente, cada parámetro de la red se adapta en cada iteración o ciclo de aprendizaje k de acuerdo con la siguiente ley:

— III.6

— — —— III.7

Aplicando la regla de la cadena para derivar el error con respecto al parámetro , se obtiene que:

—— – —— —— —— III.8

El proceso de entrenamiento de la RNA se resume en partir de un punto (aleatorio cuando no se dispone de ninguna información adicional) siendo el número de parámetros ajustables de la red, nos desplazamos en el espacio siguiendo la dirección negativa del gradiente de en el punto , alcanzando así un nuevo punto , que estará más próximo al mínimo de la función que el anterior. El proceso

continúa hasta que los parámetros dejen de sufrir cambios importantes, lo cual ocurre cuando alcanza un mínimo, es decir, cuando — para todo .

El valor , llamado también razón o tasa de aprendizaje, es el encargado de controlar cuánto se desplazan los pesos siguiendo la dirección negativa del gradiente. Este valor determina, entonces, la magnitud de dicho desplazamiento y, por lo tanto, influye en la velocidad de convergencia del algoritmo. Un valor grande favorece a una convergencia más rápida al principio, pues los cambios que se producen en los pesos permiten avanzar rápidamente en la superficie que describe la función error E. Sin embargo, en las proximidades del mínimo existe el riesgo de saltar y oscilar alrededor de él.

Se ha mencionado que el proceso de adaptar los pesos finaliza cuando — para todo , lo cual no garantiza que el mínimo alcanzado sea un mínimo global de la función . Debido a que el método usa únicamente información local sobre la superficie para determinar la dirección en cada iteración, se corre el riesgo de que los pesos se muevan hacia un mínimo, que no tiene por qué ser un mínimo global, de manera que el proceso iterativo puede finalizar en un mínimo local.

III.4 IDENTIFICACIÓN, ESTRUCTURA Y MODELO

Se presenta la estructura de un modelo adecuado para la identificación de sistemas dinámicos en un ambiente estocástico, usando redes neuronales para describir las relaciones no lineales del sistema. La información de este sub-capitulo es obtenida principalmente de (Ljung, 1999).

III.4.1 Modelo lineal

En este apartado se define vocabulario importante en el área de identificación de sistemas. El modelo se le llama lineal si puede ser representado por un modelo que tenga la forma siguiente:

III.9

Donde y son funciones de transferencia en el operador retardo de tiempo, . El operador de retraso es definido en (Norgaard et al., 2003) como:

III.10

Donde es una variable del periodo de muestreo y es una señal de ruido blanco que es independiente de las entradas.

El objetivo del proceso de identificación es determinar “buenos” estimados de las funciones de transferencia y . En donde el criterio que determina “bueno” está relacionado con la capacidad de poder predecir la salida del sistema un instante de tiempo adelante. Por lo que normalmente se le conoce a la siguiente ecuación como el predictor del modelo:

III.11

Para aclarar los términos de sistema, estructura de modelo y modelo se define la siguiente terminología:

1. Se asume que el **sistema real** es descrito por:

III.12

Con como una función de ruido blanco independiente de la señal .

2. La **estructura del modelo**, M, es un grupo de modelos candidatos parametrizados.

III.13

III.14

Donde θ denota los parámetros ajustables, y es algún un subconjunto de los donde la búsqueda por el modelo será realizada. Dentro de este trabajo solo se consideraran predicciones de un instante de tiempo adelante. Por lo que se omite de aquí en adelante para conveniencia de la notación.

III.15

Donde es el vector de parámetros y es el vector de regresión, que puede contiene entradas pasadas y salidas pasadas o señales derivadas de estas.

3. Un **modelo** es simplemente una elección particular del vector de parámetros,

.

En el caso específico de la estructura de un modelo ARX (Auto-Regression with eXternal inputs) las matrices G y H son:

III.16

III.17

Dónde:

III.18

III.19

Por lo que el modelo toma la forma de:

III.20

III.21

Con

III.22

III.23

III.4.2 Estructura no lineal basada en redes neuronales

Cuando se habla de identificación de sistemas no lineales dinámicos, el problema de seleccionar una estructura se vuelve complicado. Sin embargo, como se describe en (Ljung, 1999) una estrategia es usar las estructuras conocidas de los modelos lineales. Esta estrategia reporta múltiples ventajas como:

- Es una extensión natural de las bien conocidas estructuras de los modelos lineales.
- La estructura puede ser expandida gradualmente si se requiere una mayor flexibilidad para modelar relaciones no lineales más complejas.
- Se ajusta para el diseño de sistemas de control.

La contraparte no lineal del modelo lineal presentado en el apartado anterior es:

III.24

Donde es nuevamente el vector de regresión, mientras es el vector que contiene los parámetros ajustables en la red neuronal; conocidos como pesos. es la función realizada por la red neuronal que se asume tiene una estructura Feedforward.

Al igual que la estructura ARX, la llamada estructura NNARX por (Norgaard et al., 2003), es siempre estable porque tiene solo una relación algebraica entra la predicción y las pasadas entradas y salidas. Esto es especialmente importante en el caso no lineal por que la estabilidad es más compleja que para los sistemas lineales. Esto hace que la estructura NNARX (Figura III.6) sea una buena opción cuando el sistema es determinístico o el nivel de ruido no es significativo.

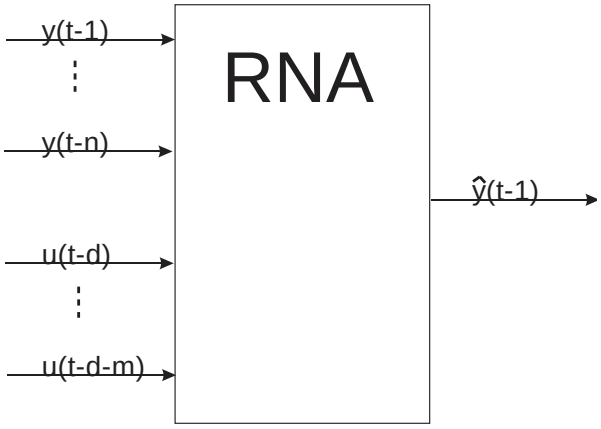


Figura III.6 Esquema de la estructura NNARX.

III.5 DISEÑO DE EXPERIMENTOS

El DoE (Diseño de experimentos por sus siglas en inglés) es usado como parte del proceso científico y una de las formas en que aprendemos cómo funcionan los procesos. Es usado para conocer el efecto de varios factores o variables de entrada sobre variables de interés en un proceso, que al ser modificados afectan su respuesta. El diseño de experimentos es una forma de ordenar la experimentación con el fin de manipular sistemáticamente las variables de entrada de un proceso para entender el efecto que estas pueden causar en la variable de respuesta (Montgomery, 2002). El DoE tiene por objetivos:

1. Determinar cuáles son los factores que tiene mayor influencia en la variable de respuesta.
2. Determinar el mejor valor de los factores controlables que influyen en la respuesta, de manera que ésta, tenga casi siempre un valor cercano al valor nominal deseado.
3. Determinar la mejor combinación de los factores controlables que reducen la variabilidad de la respuesta.
4. Establecer la combinación optima de factores controlables, con el objetivo de minimizar los efectos de las variables controlables.

El DoE posee su propio lenguaje específico, el cual es necesario conocer. Los términos más importantes se enlistan a continuación:

- Replica: Es el número de ocasiones que se efectúa una misma condición experimental en la prueba o experimento que se está haciendo.
- Aleatoriedad: Es el orden en que se ejecutan las condiciones experimentales en el experimento. La aleatoriedad cancela el efecto de factores que quizá no conocemos que están allí, incluso estos pueden estar cambiando sus niveles a medida que corremos el experimento.
- Factores: Los factores son las variables de interés para las cuales se quiere estudiar el impacto que tienen las mismas en la respuesta. Es importante saber que los factores son variables que se pueden y deben controlar durante el experimento, contrario a las variables observables que solo se pueden medir.
- Niveles: Es el número de alternativas o ajustes para cada factor.
- Variables de salida: Son las variables respuesta del experimento. La respuesta puede ser única (una sola salida de interés) o múltiple (múltiples salidas de interés). Estas pueden clasificarse en variables cualitativas y cuantitativas.

Cuando se tiene un proceso para análisis, es importante definirlo correctamente y proceder a buscar el mejor diseño de experimentos, de manera que se le pueda sacar el mejor provecho a los datos colectados por medio del análisis estadístico. Sin embargo, (Montgomery, 2002) define una metodología para la implementación exitosa de un diseño de experimentos, que de manera breve se presenta a continuación.

1. Reconocimiento y establecimiento del problema.

2. Selección de los factores y niveles de cada uno de estos.

3. Selección de la variable respuesta.

4. Determinación del diseño experimental que debe llevarse a cabo.

5. Realización del experimento para la obtención de los datos de la respuesta.

6. Análisis de los datos.

7. Conclusiones y recomendaciones.

III.5.1 ANOVA multifactorial

En un análisis de varianza (ANOVA) multifactorial se plantea si entre una cierta variable numérica continua , llamada variable respuesta, y ciertas variables categóricas llamadas factores, hay relación o no. Por ejemplo, en este trabajo es el error en la identificación de las RNA, y los factores son; el tipo de red, el número de neuronas en la capa oculta. Hay dos preguntas que, en general, se desea contestar:

1. [Factores significativos] ¿Qué factores resultan significativos?
   Se busca qué factores tienen influencia sobre la variable respuesta, que es visible en la diferencia dada por la diferencia entre los distintos niveles.

2. [Interacción] ¿La combinación de ciertos factores, posee alguna influencia en el valor de la variable respuesta?
   Nos preguntamos si la conclusión sobre el efecto que cada factor tiene sobre la variable respuesta, se mantiene independientemente de los niveles que se consideren para los factores restantes. Si sucede esto último (y por lo tanto, la respuesta a la primera pregunta es “no”), decimos que no existe interacción entre los factores; en caso contrario, que sí existe.

Si hay evidencia de que la interacción no es relevante, utilizaremos un modelo de ANOVA multifactorial sin interacción. En caso contrario, se utiliza un modelo con interacción. Este último es el más completo. A cambio, requiere en general de más observaciones que el modelo sin interacción.

(i) Modelo de ANOVA multifactorial sin interacción:

Se describe el ANOVA para una sola variable de respuesta , y dos factores , . Este puede generalizarse fácilmente para el caso en que hay más de dos factores.

Cada observación (dato) la notaremos como , entendiendo que en dicha observación el factor está en el nivel , el factor está en el nivel , y que dentro de las que tienen dichas características, nuestra observación ocupa el lugar k. Representando la media global como μ, el modelo de ANOVA sin Interacción supone que:

III.25

Donde es el efecto del factor en nivel , es el efecto del factor en nivel , y es el residuo, que entendemos debido al azar. Si se representa por a la media de todas las observaciones que tienen en nivel , y por a la media de todas las observaciones que tienen en nivel , entonces:

III.26

Para ver si es un factor significativo, se tiene la hipótesis nula y la hipótesis alternativa :

III.27

III.28

Si es rechazada, decimos que es significativo. Igualmente, para se tiene la hipótesis nula y la hipótesis alternativa :

III.29

III.30

Si es rechazada, decimos que es significativo. Además, si tiene niveles posibles, y tiene niveles posibles, entonces los requisitos o hipótesis de partida de este modelo de ANOVA son:

- Cada uno de los grupos es normal.
- (Homocedasticidad) La varianza es la misma en cada uno de los grupos.
- (Independencia de las observaciones) No hay relación entre unos datos y otros, o entre unas variables y otras.

Los requisitos anteriores se traducen en que los residuos deben ser normales y aleatorios.

(ii) Modelo de ANOVA multifactorial con interacción:

En este caso, para cada observación suponemos:

III.31

Donde , tienen el mismo sentido que en el modelo anterior, es el efecto de interacción entre en nivel , y en nivel , y es el residuo, que se entiende debido al azar. Los contrastes sobre la significatividad de Y de son análogos al modelo anterior. Aquí debemos comprobar si existe interacción entre los factores, lo cual supone contrastar la siguiente hipótesis:

III.32

III.33

, se estiman como en el modelo anterior. Sin embargo, si se escribe la media de los datos con en nivel y en nivel como , entonces se estima como:

III.34

Los requisitos del modelo son análogos a los del caso anterior.

III.6 CASO DE ESTUDIO

En este trabajo se tomara como caso de estudio un motor de corriente continua de imanes permanentes con escobillas, a continuación se da una breve descripción de las ecuaciones diferenciales y la función de transferencia en el dominio de Laplace que modelan la dinámica de este tipo de motores, describiendo posteriormente el modelo del motor utilizado.

III.6.1 Modelo matemático del motor de corriente continua

En la figura 3.1 se muestra el esquema de funcionamiento eléctrico y mecánico de un motor eléctrico de corriente continua. La velocidad del motor de corriente directa es controlada por la alimentación de un voltaje ( ) a la armadura (rotor) del motor. El voltaje alimentado produce una corriente ( ) en la armadura, que a su vez crea un campo magnético entre el rotor y el estator. El flujo magnético es constante mientras el voltaje de armadura es variable en los motores de corriente continua controlados por armadura. El campo magnético es el encargado de producir un torque, que es usado para girar el rotor que se conecta al eje del motor. En los motores de corriente continua, una fuerza contra electromotriz (FEM), que es proporcional a la velocidad del rotor, es producida en el circuito del rotor (Altintas, 2000).

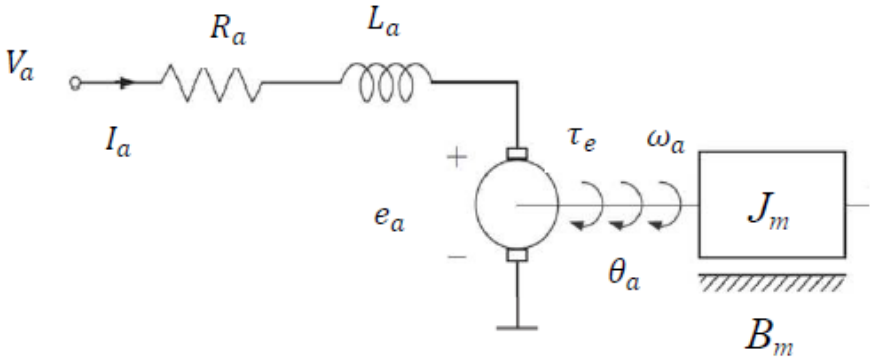


Figura III.7 Esquema eléctrico y mecánico de un motor de corriente continua de imanes permanentes.

Tabla III.2 Literales de interés para el esquema del motor de corriente continua de imanes permanentes.

Símbolo	Definición	Unidades
	Voltaje en la armadora	Volts (V)
	Corriente en la armadura	Amperes (A)
	Resistencia eléctrica	Ohms (Ω)
	Inductancia eléctrica	Henrios (H)
	Constante de fuerza contra electromotriz	(V(rad/s))
	Voltaje inducido	Volts (V)
	Torque eléctrico	Newton/metro(N/m)
	Velocidad del rotor	(Rad/s)
	Posición del rotor	Radianes (rad)

Las ecuaciones diferenciales para el circuito equivalente pueden ser construidas utilizando las leyes de Kirchhoff, en donde tenemos que:

III.35

En donde:

III.36

— III.37

III.38

Sustituyendo , y en la ecuación (III.35) y despejando —, obtenemos la siguiente ecuación diferencial de la corriente en la armadura.

— — — — III.39

Para la parte mecánica se realiza un balance de energía utilizando la ley de Newton, donde la suma de los torques debe ser igual a cero.

III.40

Donde es el torque electromagnético, es el torque generado debido a la aceleración en el rotor, es el torque generado por la velocidad en el rotor y es el torque ejercido por la carga, por lo que siendo es directamente proporcional a la corriente de armadura ( ) por una constante de torque :

III.41

Además, es directamente proporcional a la inercia del motor ( ), equivalente a una carga mecánica en el rotor, por la derivada con respecto al tiempo de la velocidad del rotor (—):

— III.42

Además, es directamente proporcional al amortiguamiento o fricción asociado a la rotación del sistema ( ) por la velocidad del rotor ( ).

III.43

Sustituyendo , y en (III.40) y despejando — , tenemos la ecuación diferencial de la velocidad de la armadura del motor de corriente continua.

— — — — III.44

Tomando las condiciones iniciales iguales a cero, la transformación de la ecuación anterior y de (III.39) al espacio de Laplace, tenemos:

— — — III.45

— — — III.46

De las anteriores ecuaciones se pueden despejar para mostrarlas de la siguiente manera:

III.47

III.48

Por lo que la función de transferencia para el motor de corriente continua, tomando el voltaje de armadura como entrada y la velocidad del rotor como salida queda de la siguiente manera:

III.49

III.6.2 Modelo del caso de estudio

Para este trabajo se va a utilizar un motor de corriente directa marca Tohoku Ricoh con un encoder óptico incremental de 400 pulsos por revolución, el número de parte del fabricante es el 52155301, se muestra en la Figura III.8.

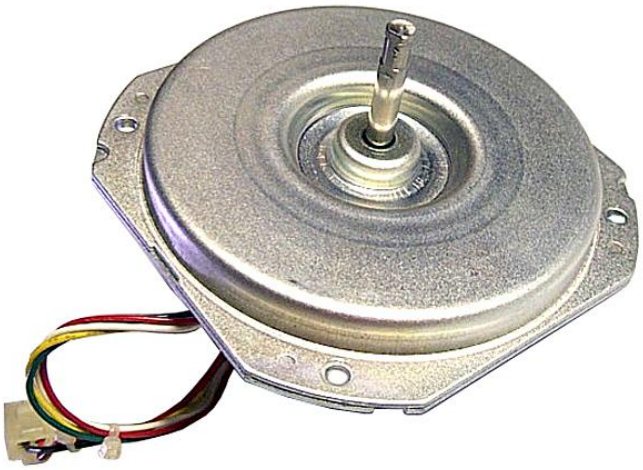


Figura III.8 Motor de corriente continua Tohuku Ricoh #52155301

Tabla III.3 Especificaciones técnicas del motor.

Voltaje de funcionamiento	24 vdc
Corriente nominal sin carga	180 ma
Velocidad sin carga	4600 rpm
Potencia nominal	55 watts
Peso	763 gr
Encoder óptico incremental	400 ppr

III.6.1 Descripción del hardware usado

Uno de los método más comunes para controlar la velocidad en un motor de corriente directa es controlando la corriente de campo, esto debido a que requiere relativamente baja potencia. Por lo que se puede controlar esta corriente por medio de circuitos electrónicos de potencia, utilizados por su habilidad para cambiar con rapidez la corriente de campo como respuesta a las señales de control.

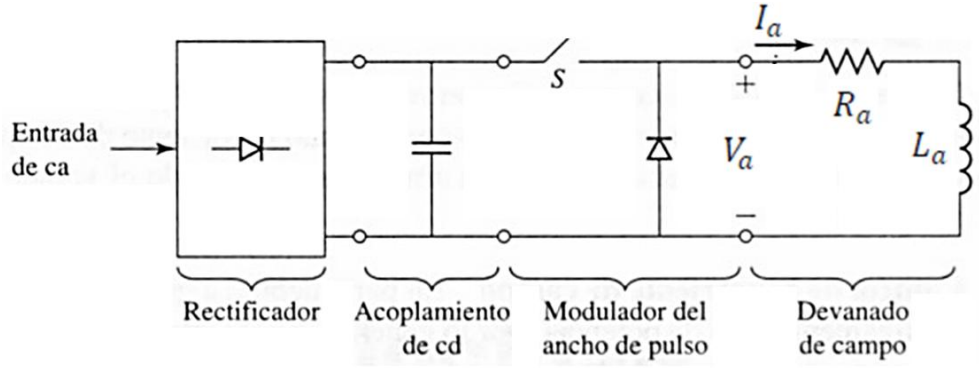


Figura III.9 Esquema de conmutación de ancho de pulso (PWM).

En este trabajo se usa el esquema de conmutación de ancho de pulso del voltaje de campo, mostrado en la Figura III.9, donde se rectifica el voltaje alterno de entrada, un capacitor de acoplamiento de corriente continua que filtra el voltaje rectificado, produciendo un voltaje de corriente directa y el modulador de ancho de pulso que consta de un solo interruptor y un diodo de operación libre. Al suponer que tanto el

diodo y el interruptor son ideales, el voltaje promedio aplicado al devanado de campo sería igual a:

III.50

En donde es el ciclo de trabajo de la onda conmutada por el PWM; es decir, es la fracción de tiempo que el interruptor está conduciendo. Como resultado la corriente promedio de campo será igual a (III.51). Por lo que la corriente de campo se regula fácilmente al controlar el ciclo de trabajo del PWM.

— — III.51

El motor cuenta con un encoder incremental de cuatrocientos pulsos por revolución con el que se aproxima la velocidad contando los pulsos sobre un base de tiempo, existe mucha literatura de cómo es que se usa las señales del encoder para este propósito por lo que no se describe a conciencia dentro de este trabajo, sin embargo se describe de modo grafico en la Figura III.10 la conexión realizada para hacer las pruebas con el motor.

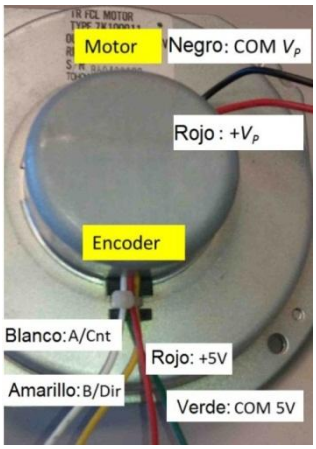


Figura III.10 Detalle de conexión del encoder.

IV METODOLOGÍA

IV.1 SIMULACIÓN Y ANÁLISIS Y DE REDES "FEEDFORWARD" VS. LRGF.

En esta sección se describe el trabajo previo necesario para la implementación de la red dentro de la FPGA. Evitando modificaciones en la etapa de implementación, ya que eso sería más complejo y resultaría en un tiempo de desarrollo mayor. Por eso en este capítulo se evalúa el funcionamiento de ambas redes neuronales antes de ir directamente a la implementación. Se destaca el diseño de experimentos realizado para optimizar el uso de recursos de la red a implementar, además de darnos información valiosa acerca del rendimiento de ambas redes durante numerosas pruebas.

Esta parte del diseño de la red neuronal va a ser desarrollado en Matlab, aprovechando las múltiples ventajas que este programa nos da, como son: Cálculos intensivos desde un punto de vista numérico, gráficos y visualización avanzada, lenguaje de alto nivel basado en vectores, arreglos y matrices, y una colección muy útil de funciones. Con su amplio rango de herramientas para modelar sistemas de control, análisis, simulación y procesamiento de prototipos, Matlab es un programa ideal para desarrollar sistemas avanzados de control, como es el caso de este trabajo.

IV.1.1 Descripción de las arquitecturas de las redes.

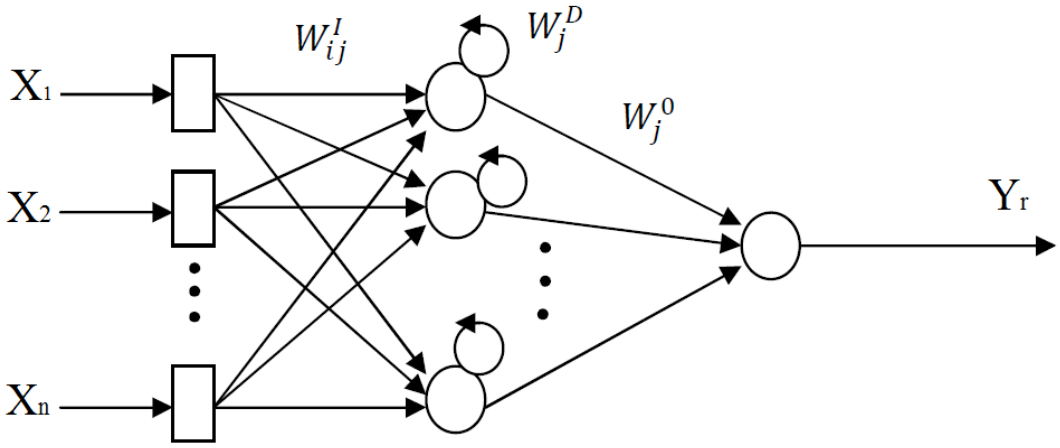


Figura IV.1Arquitectura de la red neuronal LRGF.

En este trabajo se usa una red recurrente LRGF con solo una capa oculta y un lazo de retroalimentación individual de la salida propia para cada neurona dentro de la capa oculta. Donde, son los pesos que van de la capa de entrada a la oculta, son lo pesos del lazo de realimentación y son los pesos que van de la capa oculta a la de salida (Figura IV.1). Donde la salida para la j-esima neurona de la capa oculta para una función de activación sigmoidea está dado por ( ), siendo la función de entrada de la misma ( ). Para la capa oculta la función de entrada es ( ) y la salida de la RNA está dada por la función sigmoidea ( ).

IV.1

IV.2

IV.3

IV.4

IV.5

En el caso de la red “Feedforward” La función del error que se usara en el algoritmo de aprendizaje está dada por ( ).

IV.6

Como se describe en el capítulo 3, dado que la salida de la red ( ) dependen del conjunto de pesos ajustables , el error es también función de dichos parámetros, por lo que se puede determinar el conjunto de parámetros que minimizan el error moviéndonos en la dirección negativa del gradiente de la función . Calculando las derivadas parciales de la función de error por la regla de la cadena tenemos:

IV.7

IV.8

Si se define como:

IV.9

IV.10

IV.11

Si define como:

IV.12

IV.13

IV.14

Si se define como:

IV.15

IV.16

La red LRGF se basa en la red “Feedforward”, es por esto que se puede obtener el comportamiento de la red “Feedforward” con solo definir a los pesos de la capa oculta como cero. De la misma manera el proceso de entrenamiento se puede usar en ambas, con la diferencia que en la red “Feedforward” se elimina el paso correspondiente al cálculo de los pesos . Se omite la descripción de la red “Feedforward” por ser fácilmente obtenible de la descripción ya desarrollada.

IV.17

Basándonos en el método de descenso del gradiente, cada parámetro de la red se adapta en cada iteración o ciclo de aprendizaje de acuerdo con la siguiente ley:

IV.18

El valor , llamado también razón, factor o tasa de aprendizaje, es el encargado de controlar cuánto se desplazan los pesos siguiendo la dirección negativa del gradiente.

IV.1.2 Selección de los factores y niveles para el DoE.

Como ya se ha mencionado una de las partes más complejas dentro de la aplicación de redes neuronales es la selección de aspectos específicos de su configuración,

donde se presentan múltiples posibilidades y se carece de recomendaciones definitivas que permitan seleccionar las más convenientes para cada caso.

En éste trabajo se propone el análisis de 4 factores que son conocidos por tener una importancia en el comportamiento de la RNA, estos se muestran en la

Tabla IV.1, además de mostrar el número de niveles y descripción de los niveles para cada factor.

Tabla IV.1 Factores y niveles a usar en el DoE

Factor	Niveles	Descripción
Tipo de red	2	"Feedforward" y LRGF
Parejas de entradas y salidas en tiempo pasados	4	Se usan de dos a ocho entradas de la red, con hasta cuatro retrasos en el tiempo para la entrada y salida del sistema.
Neuronas en la capa oculta	10	De una a diez neuronas en la capa oculta.
Factor de aprendizaje para el proceso de entrenamiento	8	Los valores utilizados son: [30 20 10 5 1 .1 .01 .001]

Los factores antes mencionados tienen un impacto en la capacidad de la red para identificar el sistema, en otras palabras en la capacidad de hacer el error igual a cero para cada . Es por esto que el criterio a analizar (variable de respuesta) es el error medio cuadrático (IV.19), donde se acumulan los errores en cada intervalo para los N números de muestras tomadas durante cada experimento.

– IV.19

IV.1.3 Desarrollo del DoE

Para desarrollo del DoE y el análisis posterior con la ANOVA se usara el software Minitab 15. Este programa reduce el trabajo necesario, además de brindar un sin número de gráficas que serán utilizadas durante el análisis.

Por la diversidad de niveles dentro de los factores estudiados se usa el diseño factorial de múltiples niveles, con las características descritas en la Tabla IV.2. Siendo estas características una conclusión lógica a la selección de factores y niveles antes descrita en este trabajo.

Tabla IV.2 Características del diseño factorial de múltiples niveles usado.

Factores	4
Réplicas	4
Corridas base	640
Total de corridas	2560
Bloques base	1
Total de bloques	4
Número de niveles: 2, 4, 10, 8	

Minitab se encarga de mostrar el DoE que se va a llevar acabo. Reduciendo el trabajo dentro de este programa a llenar con la información de la variable de respuesta (EMC) que se obtendrá de las simulaciones para cada uno de los experimentos que el DoE nos pide. Por lo que se procede a obtener los datos del sistema, simulando el comportamiento de ambas RNA a estos.

IV.1.4 Obtención de datos y simulaciones en PC

El propósito de la experimentación es recoger datos para representar como el sistema se comporta en algún rango de funcionamiento, la idea es variar las entradas , observando el impacto en las salidas (Figura IV.2).

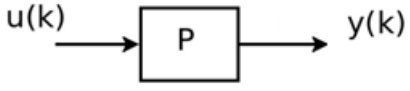


Figura IV.2 Esquema de entrada y salida del sistema.

En algunos sistemas incluso es necesario que los datos sean tomados cuando se están controlando, esto por la inestabilidad inherente del sistema. No es así para el caso de estudio de este trabajo, por lo que los datos serán obtenidos durante el funcionamiento en lazo abierto.

Otro aspecto con el que se debe tener cuidado es el diseño de la señal de entrada, que es importante si se quiere identificar el entero rango de operación del caso de estudio. Por lo que una señal de entrada que es generada como ruido blanco (Gaussiano) y filtrada en un filtro lineal es una buena recomendación (Ljung, 1999), ya que esta señal puede "virtualmente" obtener casi cualquier espectro. Un ejemplo de la señal descrita es la Figura IV.3, donde se tiene definido el ciclo de trabajo de un PWM, expresado por valores de cero a doscientos cincuenta y cuatro (byte), deseado para cada instante del experimento. Se define el valor en un byte para poder ser implementado dentro del FPGA en una tabla de consulta (LUT).

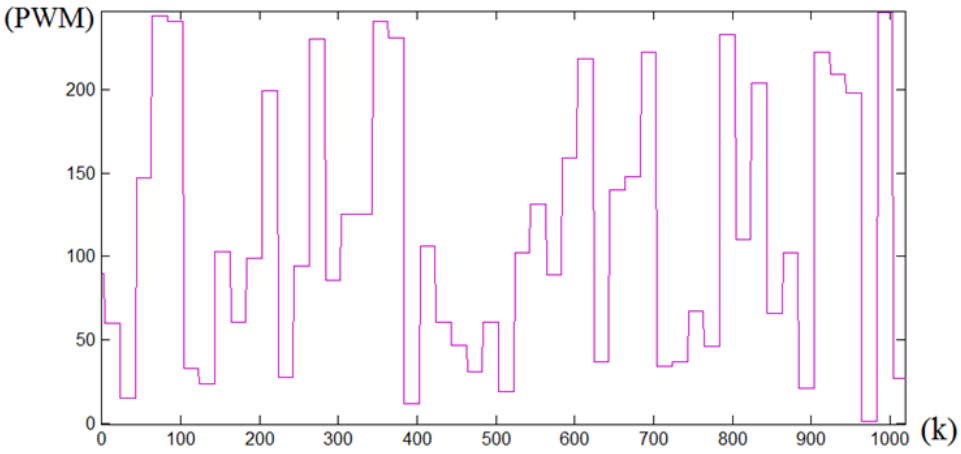


Figura IV.3 Señal PWM de entrada.

Para cualquier muestreo de datos es importante que se tenga cuidado al escoger la frecuencia del mismo, si esta se escoge demasiado alta comparada con las dinámicas de sistema se pueden tener problemas de acondicionamiento cuando se quiera identificar el sistema. Esto es importante porque una frecuencia de muestreo conduce a una mejora en el desempeño del modelo, sin embargo esto afecta el desempeño de un controlador para cerrar el lazo por que los problemas de acondicionamiento se hacen

más pronunciados (Nogard, 2003). Por lo que se usa una frecuencia de muestreo de un milisegundo que se considera adecuada para el caso particular.

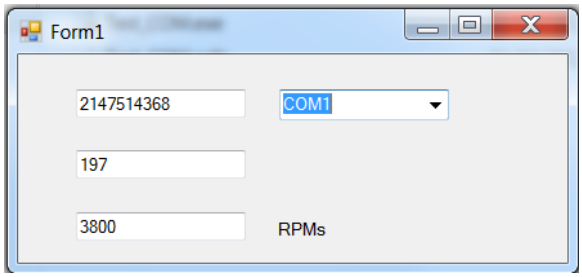


Figura IV.4 Programa para adquirir los datos desarrollado en C#.

Los datos para el diseño de experimentos fueron obtenidos usando un FPGA (Spartan-3), adquiriendo la velocidad aproximada del motor usando el algoritmo descrito por (Luna, 2011). La entrada al motor es con una señal PWM que se almacena en una LUT dentro del FPGA. Mandando ambas señales a una computadora vía comunicación serial (Figura IV.5). Dentro de la computadora se usa un sencillo programa desarrollado en C# (Figura IV.4) con el fin de almacenar, dar formato a la cadena de bytes que se reciben y almacenarlos en un archivo de texto para su manejo posterior.

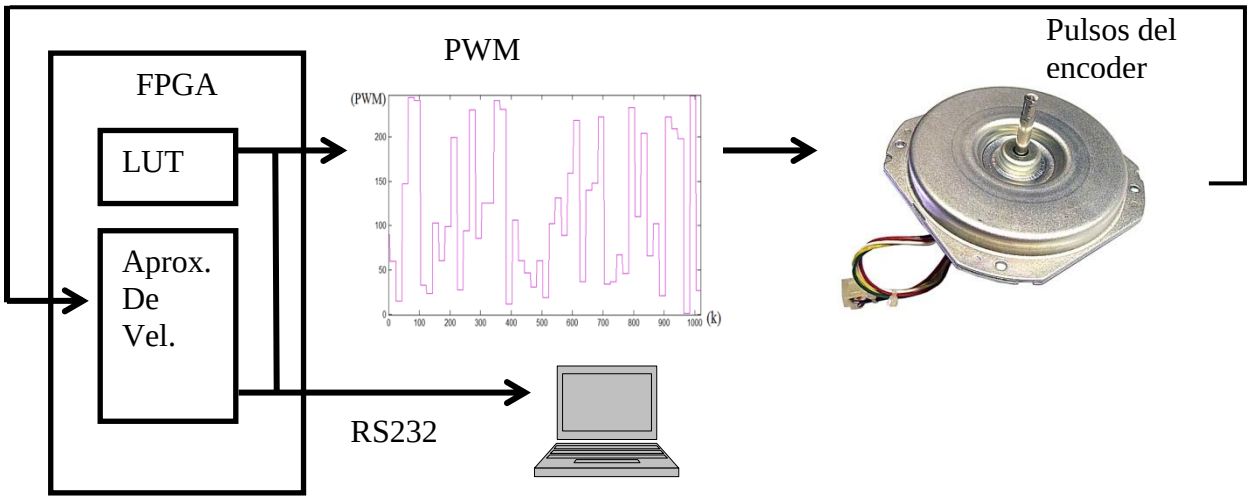


Figura IV.5 Esquema de la adquisición de datos.

Los datos obtenidos por el sistema de adquisición se escalan antes de ser usados dentro de las simulaciones, llevando a las señales a la misma varianza. Ambos señales

de los datos se miden en diferentes unidades por lo que llevaría a que la señal con mayor magnitud fuera dominante en la identificación. Se reporta también que escalar tiene beneficios como hacer el algoritmo de aproximación numéricamente más robusto y lleva a una convergencia más rápida (Polat y Yildirim, 2010).

El escalamiento de la señal antes mencionado se lleva a cabo en el programa descrito en el anexo 1. El mismo programa contiene el número de niveles para cada uno de los cuatro factores utilizados, el programa se encuentra descrito para que estos puedan ser modificados obteniendo los datos de ambas redes sin trabajo adicional. En el caso de los factores de aprendizaje, definidos dentro de un vector (FA_V), es necesario que al aumentar el número de factores de aprendizaje máximo también se definan el mismo número de constantes a ser tomadas como nuevos factor de aprendizaje dentro de los experimentos.

Este programa llama a los programas de las RNA, anexo 2 y 3, para empezar a capturar los datos de los experimentos requeridos, para luego graficar la superficie promedio (EMC) del conjunto de experimentos realizado para cada replica. El EMC para cada combinación de las simulaciones se almacena, formando a su vez matrices tridimensionales para cada red con una dimensión determinada por el número de neuronas en la capa oculta, la dimensión del vector que contiene a los factores de aprendizaje y el número de réplicas. Una gráfica por cada una de los niveles de número de entradas a la red (2, 4, 6, etc.). Estas matrices están separadas por el número de neuronas de entrada (2-12). Para poder mostrar los resultados, se promedia la respuesta de las réplicas formando un conjunto de matrices bidimensionales con una dimensión dada por (número de neuronas en la capa oculta, dimensión del vector que contiene a los factores de aprendizaje), estas matrices se muestran en la Figura IV.6, Figura IV.7, Figura IV.8, y Figura IV.9.

En el anexo 1, 2 y 3 se encuentran los programas de las RNA LRGF y Feedforward, en ambos casos los pesos iniciales son igualados a cero, con el fin de que ambas RNA partan de un mismo punto y su desempeño sea comparado en igualdad de condiciones. En ambas casos el programa se separa en cinco bloques, la inicialización de pesos y variables a cero, la actualización de las entradas a la RNA, el cálculo de la

salida de la red, cálculo del algoritmo de retropropagación dinámica y la actualización de pesos. Los últimos cuatro bloques entran en un ciclo finalizado por lo largo del experimento, en este caso definido a 1024 muestras. Una vez finalizado los bloques de la red se calcula el EMC para el experimento.

Es importante aclarar que la finalidad del trabajo previo a la implementación es indicar cuál es la RNA con mejor desempeño y ayudarnos a tomar las decisiones correspondientes a la selección de la configuración más adecuada de la RNA a implementarse en el FPGA, por lo que se usa la ANOVA multifactorial con interacciones (Tabla V.1) descrita en el apartado III.5.

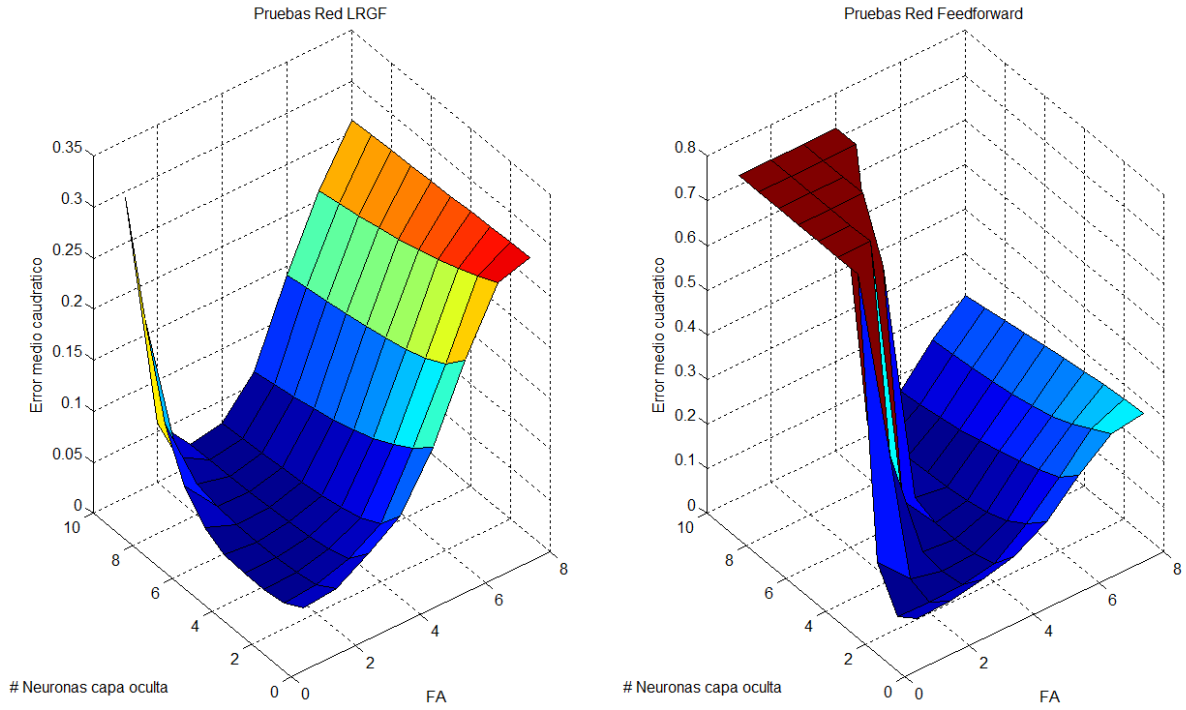


Figura IV.6 Datos para dos entradas de la red.

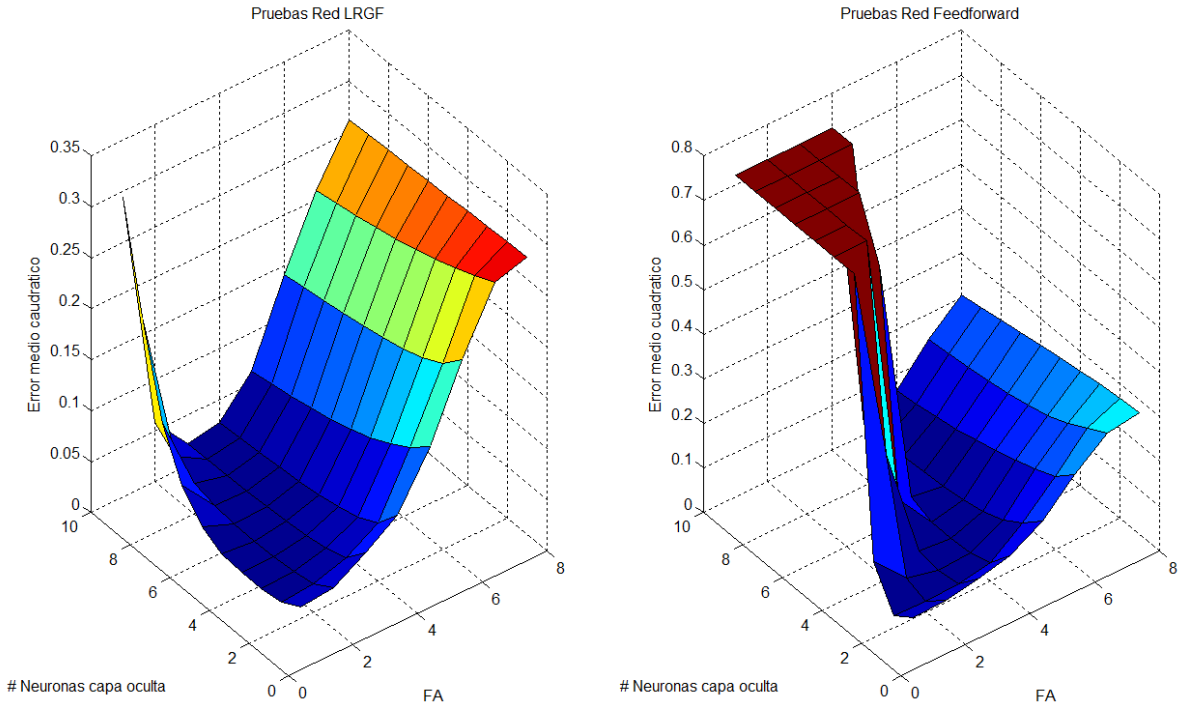


Figura IV.7 Datos para cuatro entradas de la red.

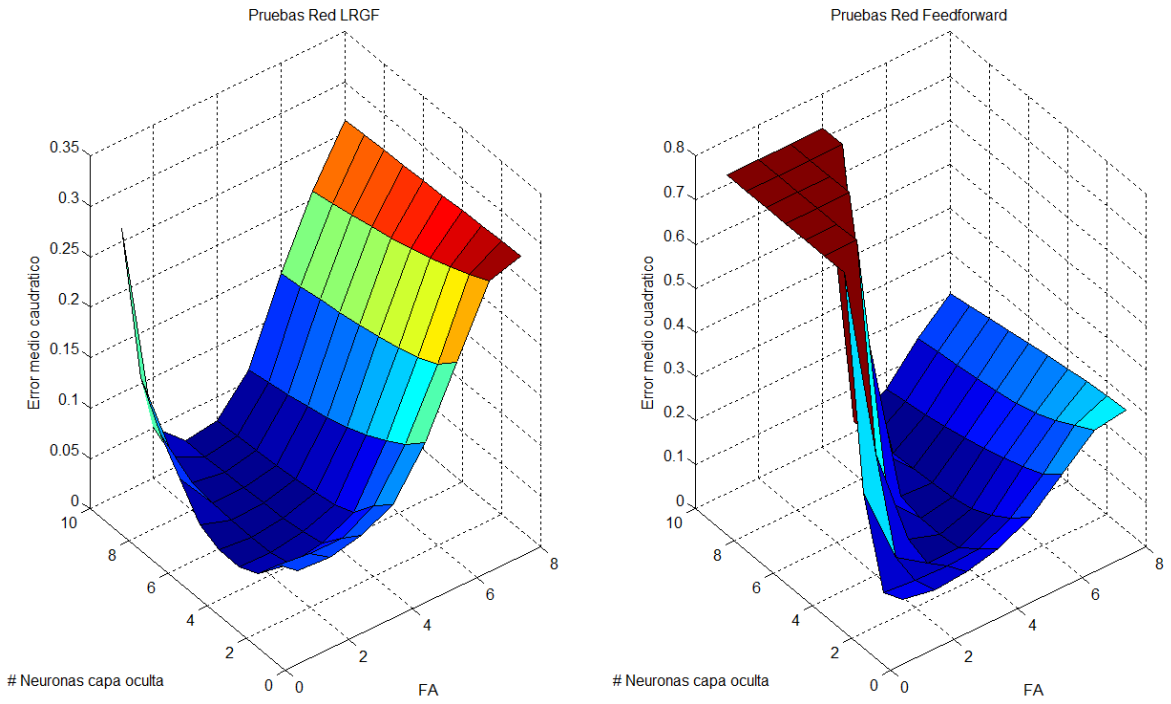


Figura IV.8 Datos para seis entradas de la red.

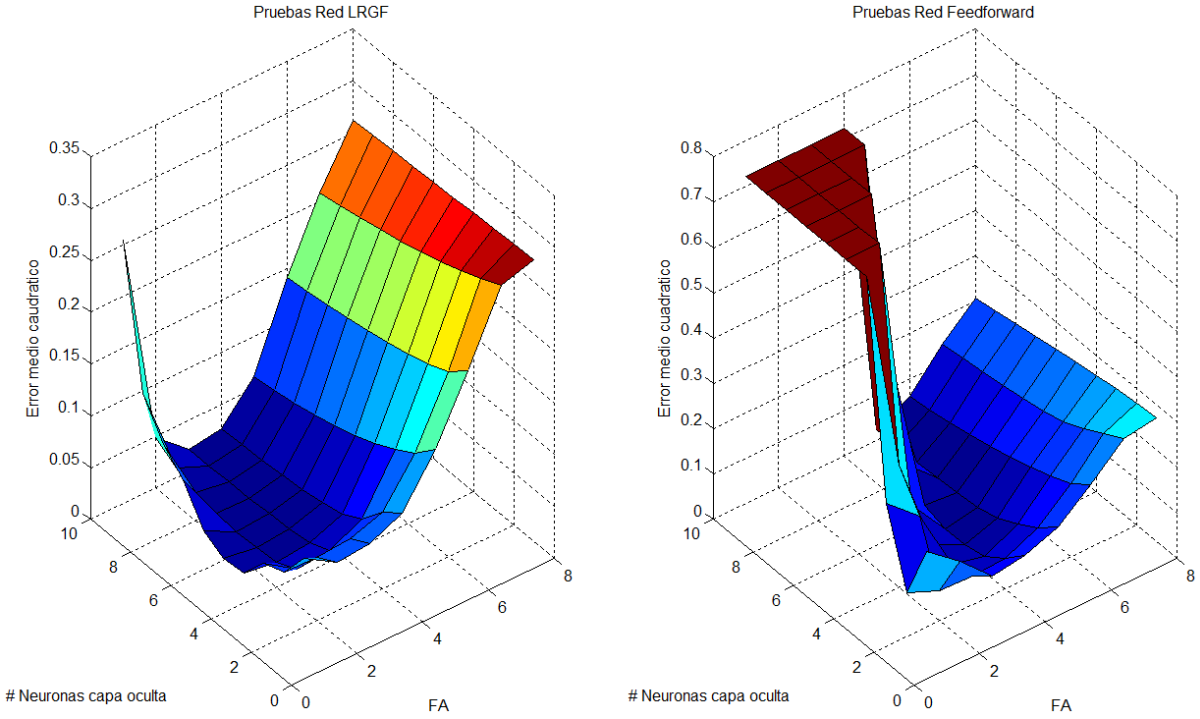


Figura IV.9 Datos para ocho entradas de la red.

IV.2 DISEÑO EN DE LA RNA LRGF EN FPGA

En este apartado se describe a detalle la implementación de la RNA en FPGA, el orden se toma de los bloques de menor jerarquía hasta llegar a la RNA completa. Es importante repasar el apartado de descripción de la RNA.

Es importante destacar que todas las ecuaciones tomadas para la implementación fueron sometidas a una revisión por parte del diseñador, como resultado se juega con el orden de los coeficientes en ambas ecuaciones. Esto con el objetivo de reducir problemas con el manejo de formatos en punto fijo y un exceso en el uso de operaciones. Sin embargo, esto puede llegar a confundir al lector. Se recomienda poner especial atención a los esquemas junto con las ecuaciones correspondientes, encontradas en el apartado IV.1.1, para no tener problemas en entender los diseños en VHDL.

IV.2.1 Neurona

La Figura IV.10 muestra el esquema de funcionamiento general de la neurona, compuesta principalmente por una unidad MAC y una tabla de consulta. La unidad MAC recibe una pareja de datos (entrada y peso) por cada ciclo de tiempo, siendo controlada la cantidad de estos por la máquina de estado correspondiente a la neurona. La tabla de consulta de la función sigmoidea se evalúa desde -7 hasta 7 en un formato de punto fijo signado de 4.8, con una resolución para el argumento de 0.00341797. Se agrega un bloque para evitar que alguno de los resultados de la unidad MAC supere el rango donde se evalúa la tabla de consulta, evitando problemas de formato en el argumento de la tabla de consulta. La salida de la función sigmoidea es como tal la salida de la neurona, con un resultado en un rango de 0 a +1, con un formato de punto fijo a 1.12.

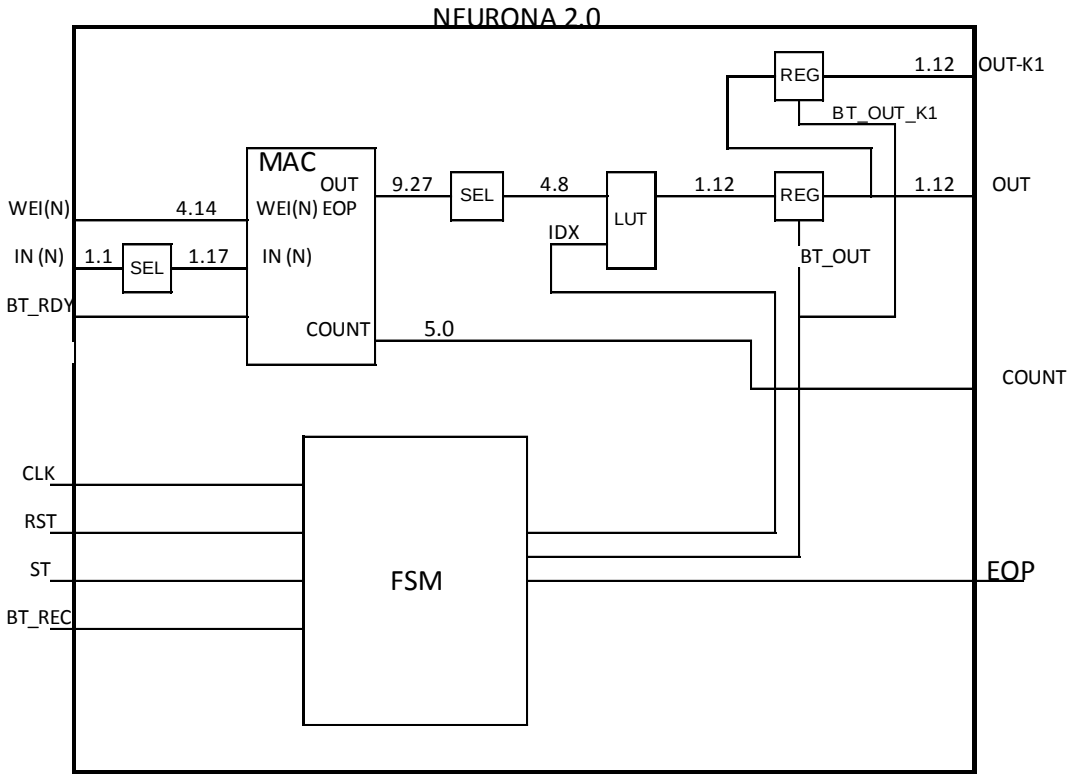


Figura IV.10 Esquema de la neurona para la red LRGF.

La neurona es la unidad de procesamiento de la red, como se discutió anteriormente esta se implementa con una retroalimentación en la capa oculta. Por lo que la neurona se configura con un bit “BT_REC”, creando un ciclo más para que la MAC agregue la retroalimentación. Además, este bit habilita o deshabilita el registros de salida “OUT-K1”, por lo que la neurona puede almacenar la salida adicional necesaria de la RNA escogida. La señal “COUNT” sirve para sincronizar la entrada de los datos al función multiplica y acumula de la neurona. En Figura IV.11 se encuentra la máquina de estados para la neurona.

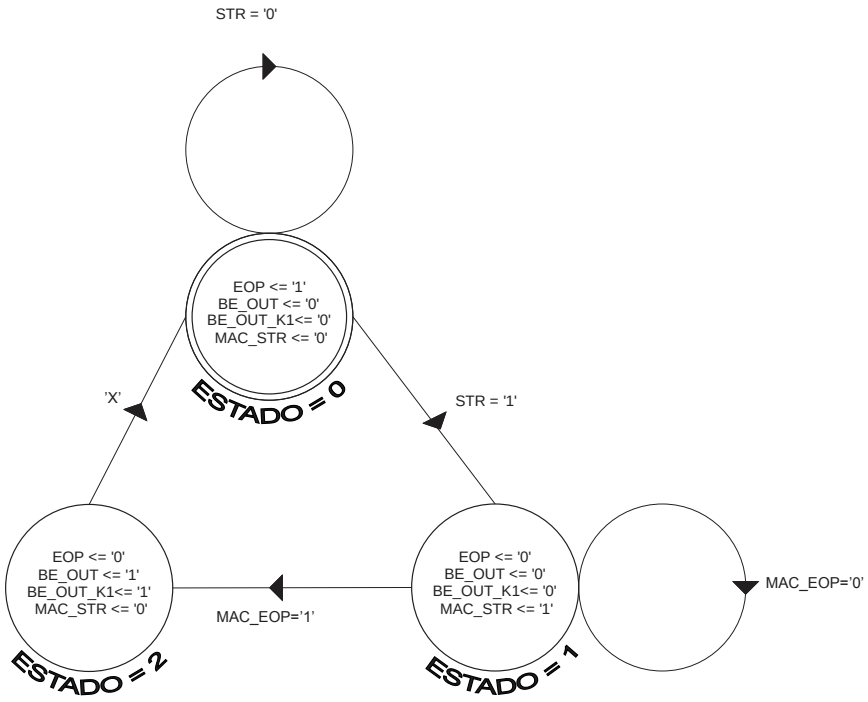


Figura IV.11 Máquina de estado del diseño de la neurona.

IV.2.2 Lógica de pesos

En la Figura IV.12 se presenta el esquema de funcionamiento de la lógica de pesos utilizada en este trabajo. Cada uno de los pesos utilizados dentro de la red es almacenado y actualizado en este bloque. Es bien sabido que dentro de la teoría de las RNA la modificación de pesos con el algoritmo de aprendizaje es parte fundamental, además de la necesidad de inicializar a los pesos y variables correspondientes al algoritmo de aprendizaje. Por lo que se utiliza un multiplexor para poder seleccionar entre un peso inicial “Wi” y los pesos futuros calculados por el algoritmo de aprendizaje “Wk+1”, controlando la actualización del mismo con la señal bit “Act”.

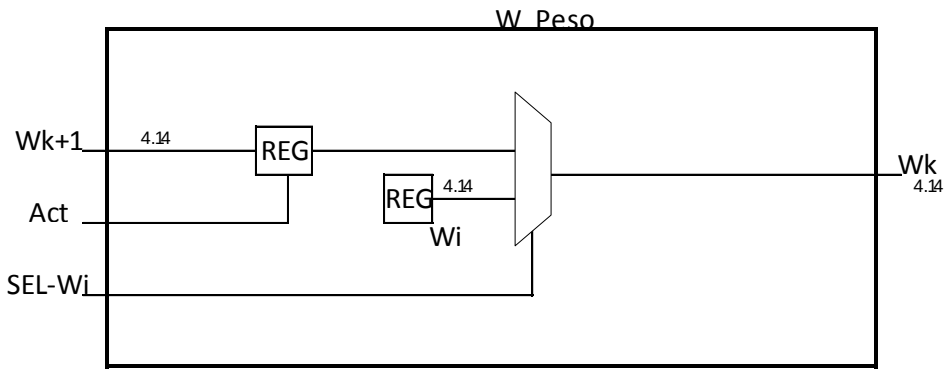


Figura IV.12 Esquema de la lógica de pesos.

IV.2.3 Algoritmo de aprendizaje

A continuación se describe el diseño de lógica requerido para la actualización de pesos dentro del FPGA. Para ver a detalle las ecuaciones del algoritmo de aprendizaje, revisar el apartado IV.1.1 de descripción de la RNA usada en este trabajo.

Por ser en su mayoría lógica combinacional, los siguientes diseños no cuentan con máquina de estado. Los registros encontrados dentro de los siguientes diseños serán activados por la máquina de estado del siguiente nivel de jerarquía en el diseño.

(i) Calculo de pesos en la capa de salida

Para el cálculo del peso en la capa de salida, donde puede ser cualquier número entre 1 y el número de neuronas en la capa oculta. Se tienen dos diseños:

El primero (Figura IV.13) encargado de calcular el coeficiente α(k), descrita en la ecuación VI.9, que solo tiene como entradas la salida actual y la salida deseada del sistema. Es importante observar que la ecuación VI.9 contiene a la diferencia entre la salida de la red LFGR y la velocidad del motor de corriente directa, esto permite que el termino α(k) y el error de aproximación pueda ser reutilizado en bloques de diseño posteriores.

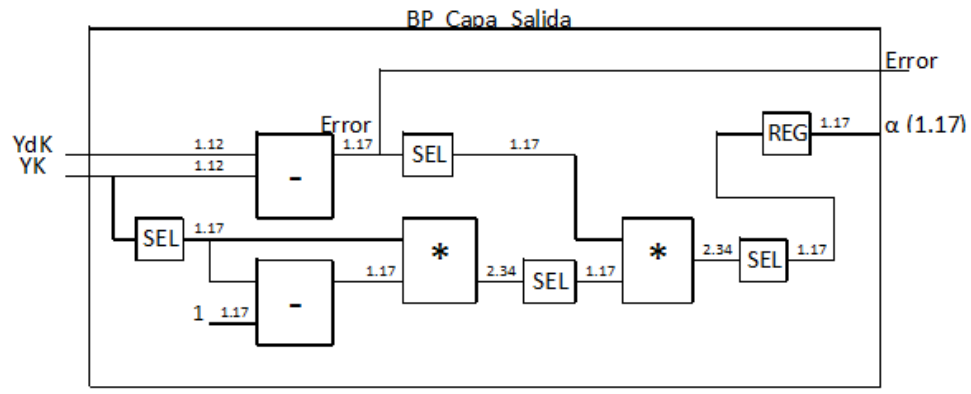


Figura IV.13 Esquema del bloque de cálculo del coeficiente α y el error en la capa de salida.

El segundo (Figura IV.14) es el encargado de calcular el valor del peso del siguiente ciclo de operación de la RNA en la capa de salida. Este bloque utiliza como entrada el cálculo de α de la ecuación IV.9, el factor de aprendizaje de la RNA, el peso actual y la salida de la j-esima neurona de la capa oculta. El funcionamiento de este bloque es obtenido de la ecuación VI.10 y La ley de actualización de pesos encontrada en la ecuación VI.18.

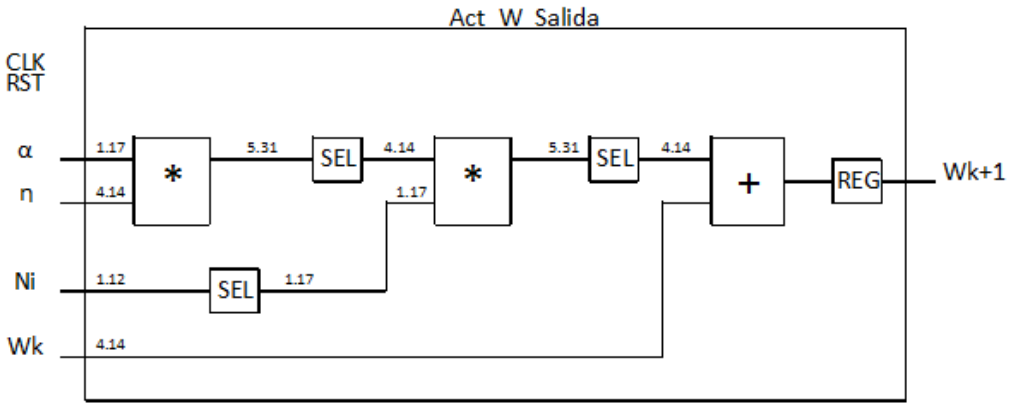


Figura IV.14 Esquema de la actualización de pesos en la capa de salida.

(ii) Calculo de pesos en la capa oculta y la neurona recurrente

Para el cálculo de los pesos de la capa de entrada , y los pesos de la capa oculta se tienen los siguientes tres diseños:

El primero (Figura IV.15) se encarga de encontrar el j-esimo coeficiente (ecuación IV.12). En el mismo diseño se almacena el coeficiente para ser utilizado en bloques posteriores.

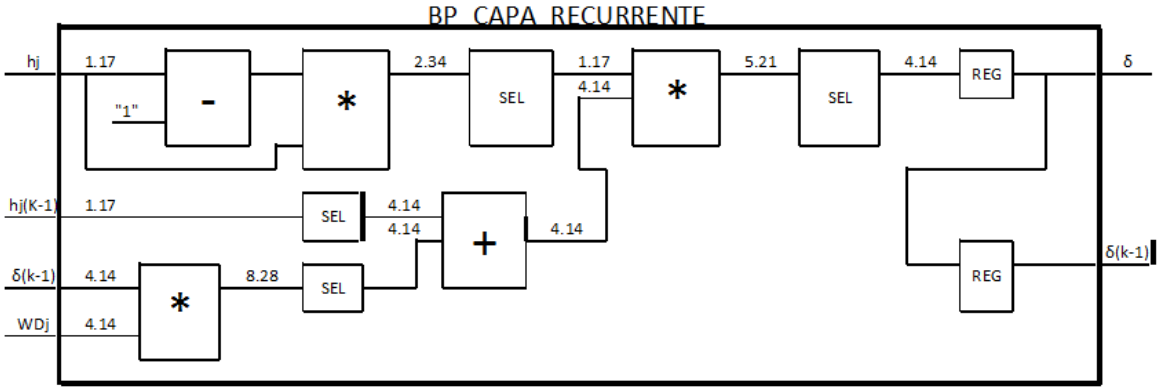


Figura IV.15 Esquema del cálculo de los coeficientes y en la capa oculta.

El segundo (Figura IV.16) se encarga del cálculo de solo uno de los coeficientes de la matriz (ecuación IV.15), donde el número de diseños necesarios en paralelo depende del número de neuronas en la entrada de la red. En el mismo diseño se almacena el coeficiente para ser utilizado en bloques posteriores.

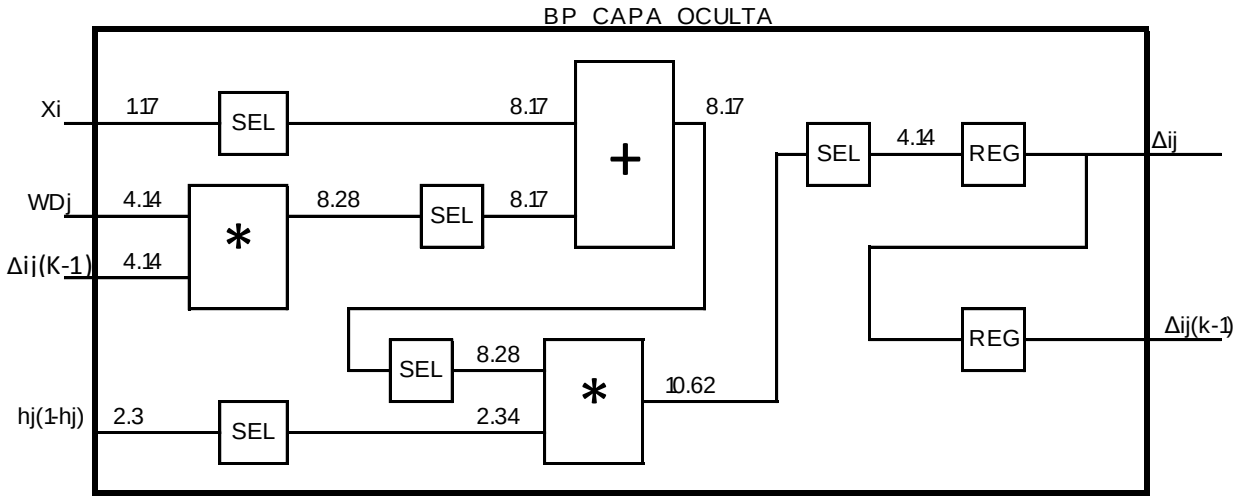


Figura IV.16 Esquema del cálculo de los coeficientes Δ_ij (k) y Δ_ij (k-1) en la capa oculta.

El tercero (Figura IV.17) es el encargado de calcular los valores de los pesos y del siguiente ciclo de operación de la RNA en la capa oculta y de entrada respectivamente. Este bloque utiliza como entrada los coeficientes , y calculados en diseños anteriores, por lo que también utiliza a los pesos de la capa de entrada , los pesos de la capa oculta y el factor de aprendizaje de la RNA. Este bloque depende del número de neuronas en la capa de entrada, ya que al crecer genera un mayor número de pesos en la capa de entrada ( ) a ser actualizados. Es por esto que la parte del diseño en el recuadro punteado debe ser duplicada y agregada en el diseño por cada neurona en la capa de entrada que tengamos. El funcionamiento de este bloque es obtenido de la ecuación VI.13, la ecuación VI.16 y la ley de actualización de pesos encontrada en la ecuación VI.18.

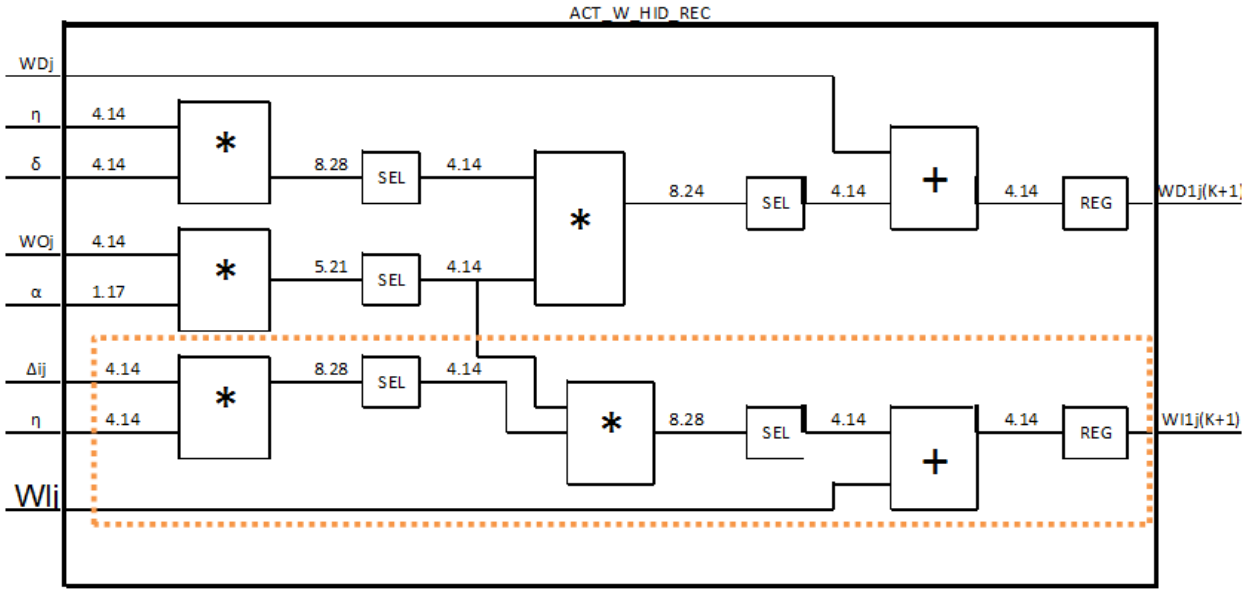


Figura IV.17 Esquema de la actualización de pesos en la capa oculta.

IV.2.4 Diseños de las capa oculta y de salida de la RNA LRGF

Las capas de la red LRGF se construyen con los diseños ya descritos, sin embargo, las capas dependen totalmente de la configuración de la red. Es por esto que los esquemas siguientes son específicamente para una configuración 2-7-1 de la red LRGF. Esta configuración es la que se toma como la óptima por los resultados obtenidos del DoE, para un mayor detalle revisar el capítulo V.

(i) Capa de Salida

La capa de salida es mostrada en la Figura IV.18, esta recibe como entrada las salidas de las neuronas en la capa oculta y la señal de velocidad actual. Como salidas tiene el coeficiente α, el error de aproximación de la identificación, la salida de la red LRGF y cada uno de los pesos de la capa de salida ( ). La capa de salida cuenta con solo una neurona, por lo que el diseño solo depende del número de neuronas en la capa oculta. El diseño obtenido se puede adaptar a una modificación de este número de neuronas con facilidad. Además de modificar el valor del número de

neuronas en la capa oculta en la señal “CYC” al correcto, basta con agregar la parte del diseño en el recuadro punteado en la Figura IV.18 y asignarle las entradas correctas. Dado que el factor de aprendizaje (η) y el coeficiente (α) son igualmente utilizados para los bloques adicionales, solo se tiene cuidado de direccionar la salidas de las neuronas y sus bloques de pesos correspondientes. En caso de reducir el número de neuronas en la capa de oculta se debe realizar el mismo procedimiento.

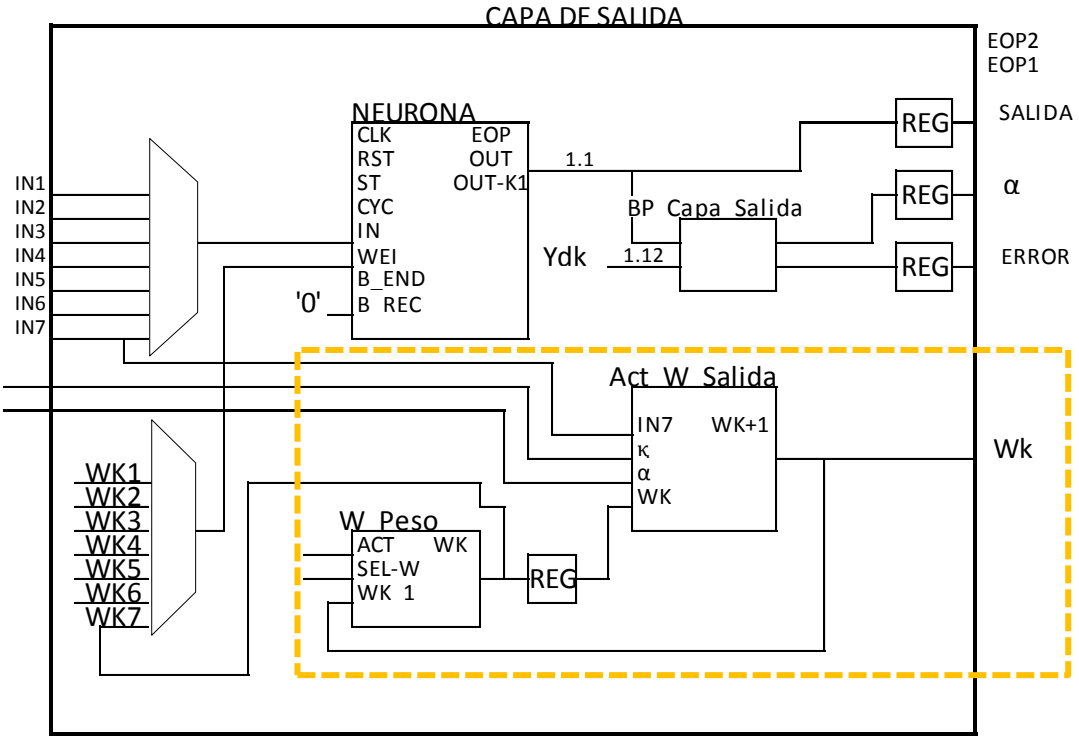


Figura IV.18 Esquema de la capa de salida.

Como ya se mencionó se tienen 7 entradas a la capa de salida, que son las salidas de las neuronas en la capa oculta. Por lo que ese es el número de ciclos de reloj a ser realizado por la MAC dentro de la neurona para poder darnos una salida, una vez que los ciclos se han terminado y se evalúa la salida de la MAC en la tabla de búsqueda, la neurona levanta la bandera “EOP”. Como se puede ver en la máquina de estados en la Figura IV.19, con esta bandera se empieza con el cálculo del error de aproximación y el coeficiente α. En este punto se habilitan los registros de salida para que sean

guardados los datos de ese ciclo. Una vez guardado los datos se da paso a la actualización de los pesos. De manera general en el primer ciclo de la RNA se necesita un conjunto de pesos iniciales, es por esto que la máquina de estados separa el primer ciclo de los demás con el uso de una señal llamada “W_sel”. Esta señal llega al bloque de pesos y controla que el peso inicial solo sea usado durante el primer ciclo.

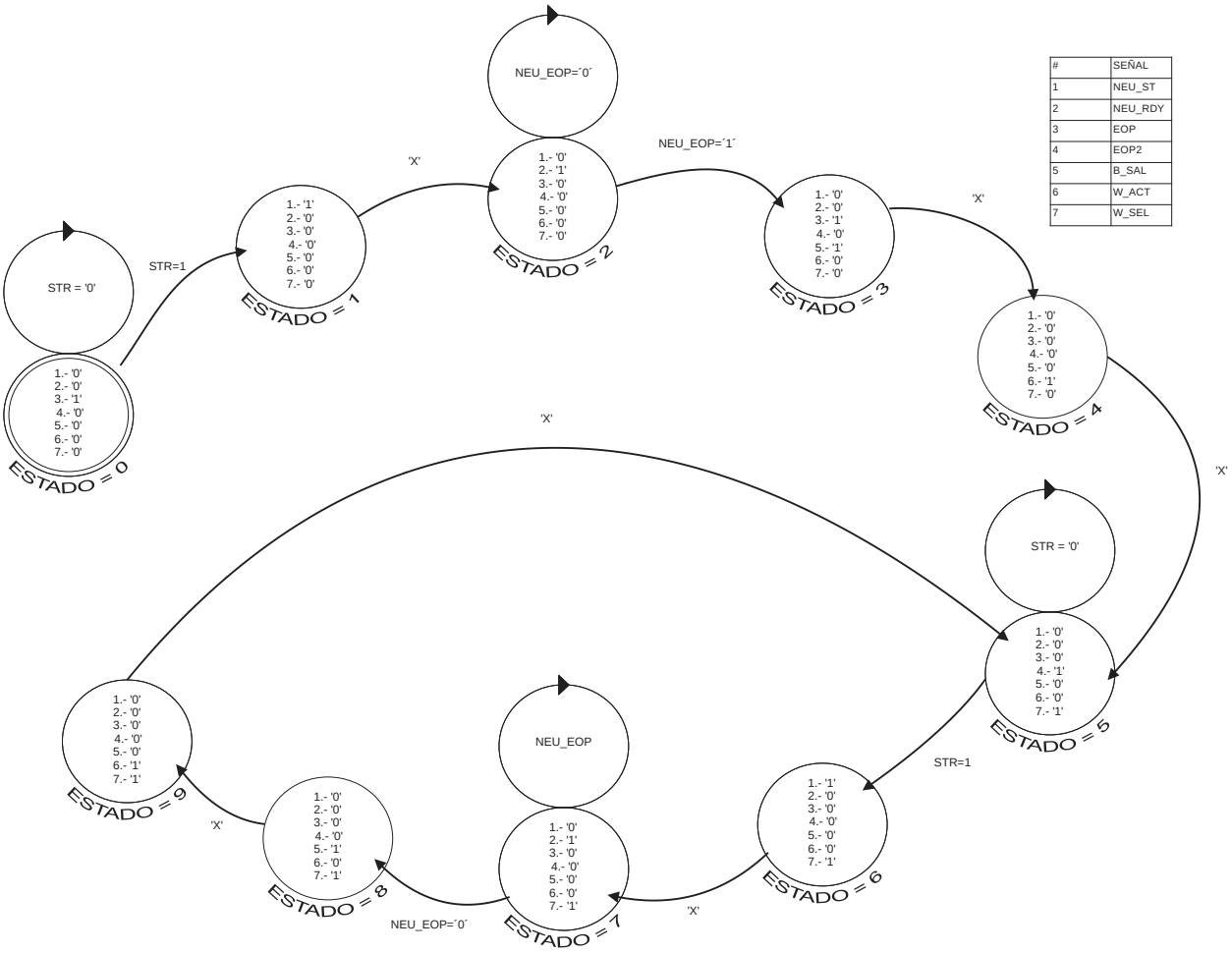


Figura IV.19 Máquina de estados de la capa de salida.

(ii) Capa oculta

La capa oculta del diseño contiene la entrada de información a la red, es decir, también contiene a la capa de entrada de la RNA. Esta recibe la información del sistema a identificar, el factor de aprendizaje, el coeficiente α y los pesos de la capa de

salida . Como salida de la capa oculta se tiene la salida de cada una de las neuronas contenidas en esta y las señales necesarias para coordinar el funcionamiento de ambas capas.

En la Figura IV.20 se puede observar la unidad base para formar la capa de salida. Esta unidad es llamada base por qué sirve como plataforma para armar la configuración deseada de la RNA a partir de los bloques correspondientes a esta capa descritos anteriormente. El número de neuronas en la capa oculta determinara cuantos de los bloques de la Figura IV.20 trabajaran de forma paralela en el diseño final. Este esquema muestra la construcción específica para una configuración de neuronas de 2-7-1, donde los bloques en los recuadros dependen del número de entradas a la RNA, 2 entradas. Como ya se discutió en apartados anteriores, algunos bloques pueden sufrir pequeñas modificaciones internas para poder adaptar el diseño de la RNA.

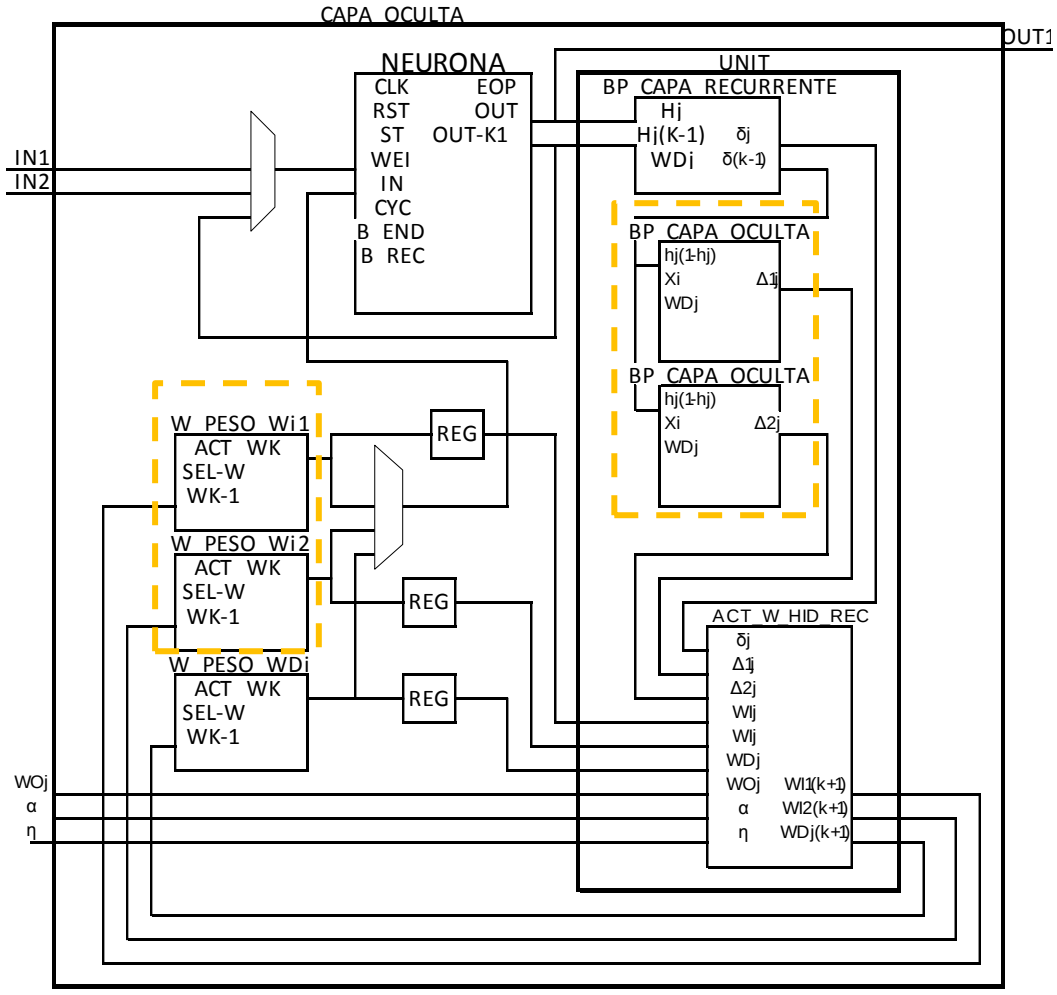


Figura IV.20 Esquema de la base para la capa oculta.

Como se observa en la máquina de estados de la capa oculta (Figura IV.21) la capa oculta trabaja de forma similar a la capa de salida, inicia con una señal llamada “STR” que habilita la neurona, para la capa oculta el número de ciclos a realizar por la unidad MAC es el número de entradas a la red más el ciclo que se agrega de la recurrencia, cuando la neurona tiene lista la salida levanta una bandera “NEU_EOP” . En este punto es donde la diferencia entre ambas capas se acentúa, la capa oculta tiene que esperar a que la capa de salida. La capa oculta necesita los valores del coeficiente α calculados por la capa de salida para poder calcular sus pesos. Esto se resuelve dejando que la capa oculta calcule las salidas de las neuronas, que pasan a la capa de salida, quedando en espera de la bandera de final del ciclo de la capa de salida “OUTL_EOP” para empezar a calcular los coeficientes necesarios para actualizar los pesos. Como

en la capa de salida se sigue separando el primer ciclo de la RNA para poder usar los pesos iniciales adecuados.

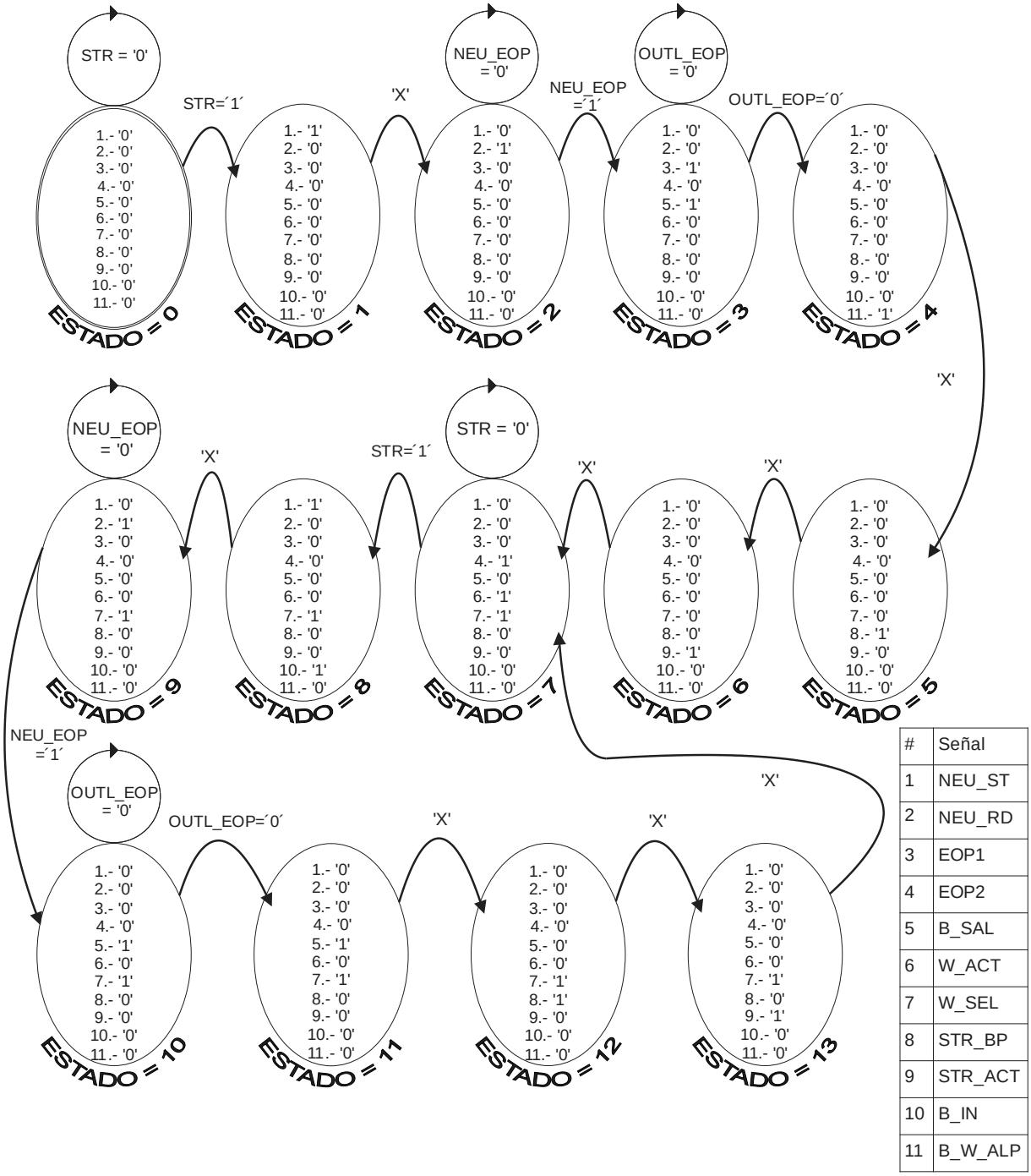


Figura IV.21 Máquina de estados de la capa oculta.

IV.2.1 Diseño final de la RNA

En este apartado se presenta el diseño final de la red LRGF en VHDL, la red se presenta en la Figura IV.23 y la máquina de estado en la Figura IV.22. La red se basa principalmente en la capa oculta y de salida, descritas anteriormente. La máquina de estados se limita a iniciar la capa oculta para que termine el primer paso de calcular las salidas de las neuronas, una vez terminado este paso inicia la capa de salida. La capa de salida calcula la salida y los pesos del siguiente ciclo de funcionamiento antes de levantar la bandera de “OUT_EOP2”, una vez que se detecta esta bandera se procede a la segunda etapa de la capa oculta, donde calcula los pesos del siguiente ciclo de operación, indicando el término de este con la bandera “HID_EOP2”. Se utiliza un contador incremental y una compuerta AND para poder controlar la cantidad de ciclos a ser ejecutados por la red neuronal (Figura IV.23)

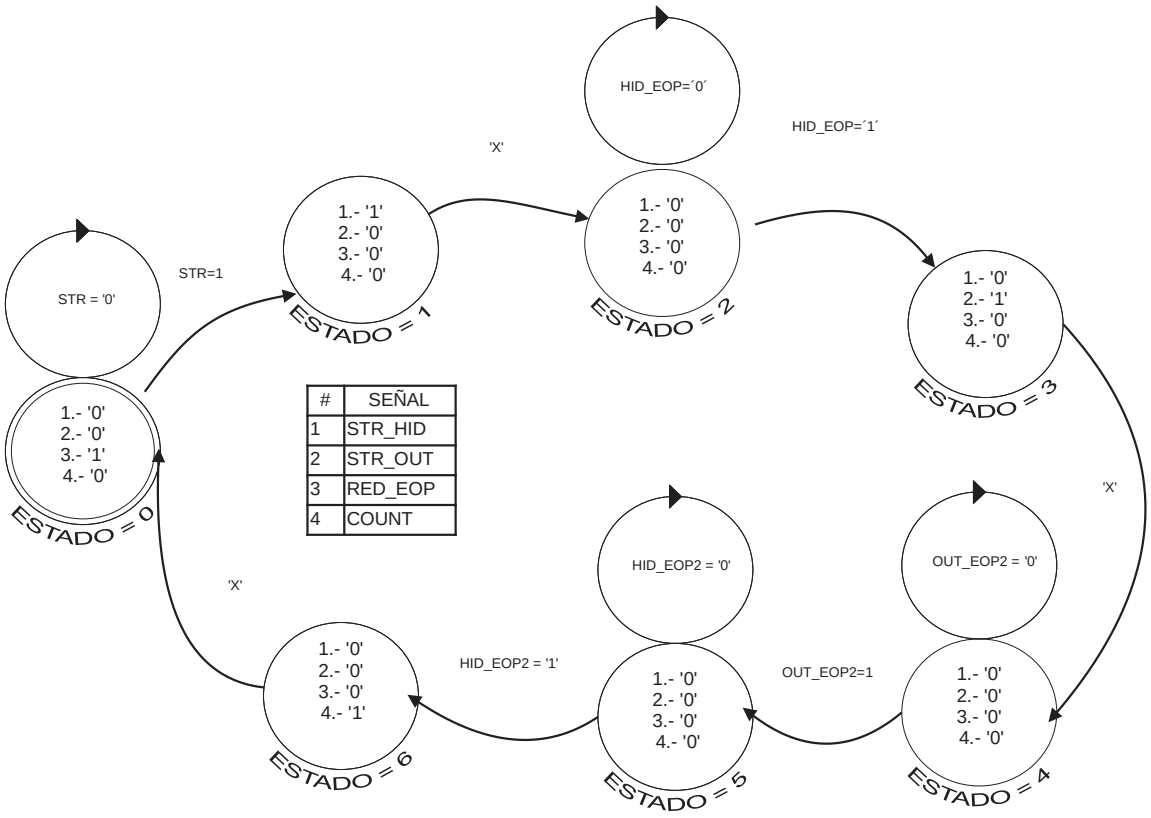


Figura IV.22 Máquina de estados de la RNA.

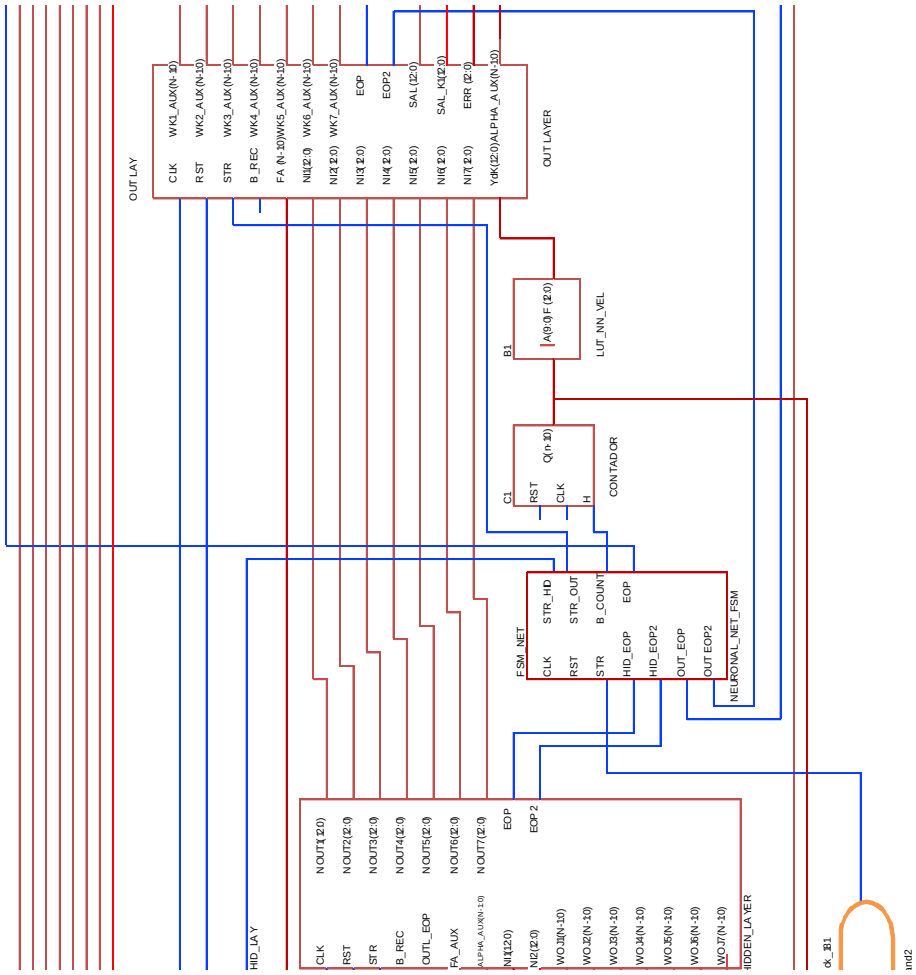


Figura IV.23 Esquema del diseño de la RNA LRGF.

V RESULTADOS Y CONCLUSIONES

V.1 RESULTADOS

Se presenta los resultados obtenidos del desarrollo del DoE, los datos obtenidos se encuentran representados por el conjunto de figuras (Figura IV.6, Figura IV.7, Figura IV.8, Figura IV.9) que contienen simulaciones de corridas de identificación realizadas en Matlab por ambas RNA con los datos obtenidos del sistema físico.

Tabla V.1 Análisis de varianza para EMC, utilizando SC ajustada.

Fuente	GL	SC sec.	SC ajust.	MC ajust.	F
RNA	1	15.9402	15.9402	15.9402	930519.45
Entrada	3	0.0074	0.0074	0.0025	144.02
Capa oculta	9	7.0880	7.0880	0.7876	45973.64
FA	7	14.4642	14.4642	2.0663	120622.41
RNA*Entrada	3	0.0027	0.0027	0.0009	52.22
RNA*Capa oculta	9	8.2074	8.2074	0.9119	53234.70
RNA*FA	7	29.1308	29.1308	4.1615	242931.87
Entrada*Capa oculta	27	0.0620	0.0620	0.0023	133.95
Entrada*FA	21	0.0305	0.0305	0.0015	84.81
Capa oculta*FA	63	20.8627	20.8627	0.3312	19331.28
RNA*Entrada*Capa oculta	27	0.0300	0.0300	0.0011	64.78
RNA*Entrada*FA	21	0.0256	0.0256	0.0012	71.23
RNA*Capa oculta*FA	63	19.2166	19.2166	0.3050	17805.99
Entrada*Capa oculta*FA	189	0.1761	0.1761	0.0009	54.38
RNA*Entrada*Capa oculta*FA	189	0.1203	0.1203	0.0006	37.14
Error	1920	0.0329	0.0329	0.0000	
Total	2559	115.3973			
S = 0.00413890 R-cuad. = 99.97% R-cuad.(ajustado) = 99.96%					

A pesar del buen resultado en describir el fenómeno ( ) del EMC con los factores escogidos, se tiene un número de alto de datos atípicos reportados por Minitab. Este es uno de los principales problemas en el uso de la ANOVA, ya que la ANOVA no es capaz de modelar al sistema para el rango de los valores escogidos. En este caso con la gráfica de interacciones del sistema (Figura V.2) se observa un comportamiento atípico de la RNA Fedforward para valores de factores de aprendizaje mayores a 1, además de que un número superior a 3 neuronas resulta en un mal desempeño de la RNA “Feedforward” en la identificación. Este comportamiento de la red es la principal causa de la generación de datos atípicos en el DoE, y aunque es

posible plantear un DoE sin esta RNA para eliminar los datos atípicos se considera innecesario. La Grafica de efectos principales (Figura V.1) e interacción de factores (Figura V.2) nos muestran el comportamiento de ambas redes para cada uno de los niveles de los factores propuestos.

Como se observa en el conjunto de figuras (Figura IV.6, Figura IV.7, Figura IV.8, Figura IV.9), se tiene una cantidad importante de información pero es difícil de leer de este modo, por lo que se utiliza la gráfica de efectos principales que muestra la media para todos los experimentos de un nivel y factor especifico, sirviendo como un indicador del desempeño de cada uno de los niveles para todos los factores. Esta gráfica se muestra en la Figura V.1.

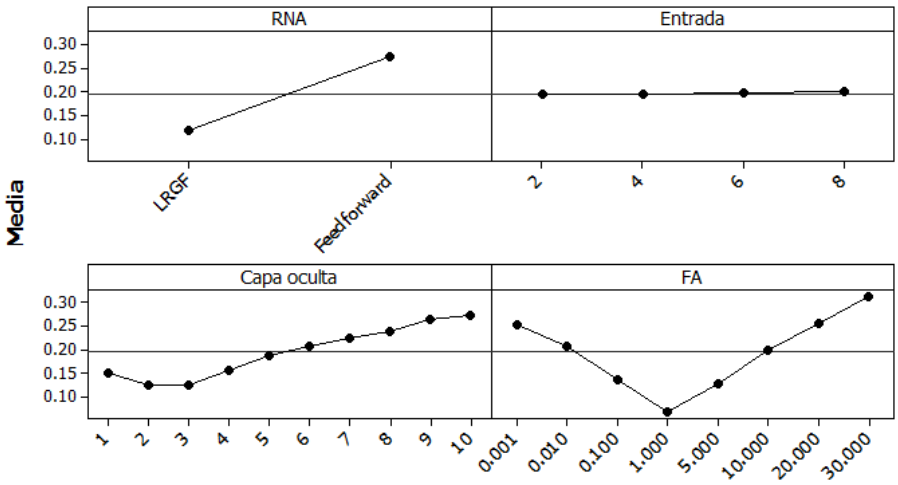


Figura V.1 Gráfica de efectos principales para el EMC con cuatro factores.

De la gráfica de efectos principales se obtienen múltiples resultados:

- El número de neuronas en la capa de entrada es un factor que se propuso y no afecta significativamente el rendimiento de la red es el número de entradas a la red. Al ser la media de experimentos similar para dos, cuatro, seis y ocho entradas, se eligen dos neuronas por ser el número más bajo de entradas a la red para la implementación, ya que aunque no tiene un efecto en el desempeño, si afecta la cantidad de lógica requerida dentro del FPGA.

- La red LRGF se desempeñó mejor que la Feedforward en cuanto a promedio de todas las pruebas. También se observa claramente en los graficas de superficie al comparar el rango de valores que toma el EMC.
- En el caso del factor de aprendizaje y de las neuronas en la capa oculta se observa que el menor EMC se encuentra en niveles centrales, sin embargo se debe tener cuidado porque se está hablando de la media para todos los experimentos. Lo que causa que ambas redes afecten a la Figura V.1 lo que oculta los verdaderos valores óptimos para cada una de las redes.

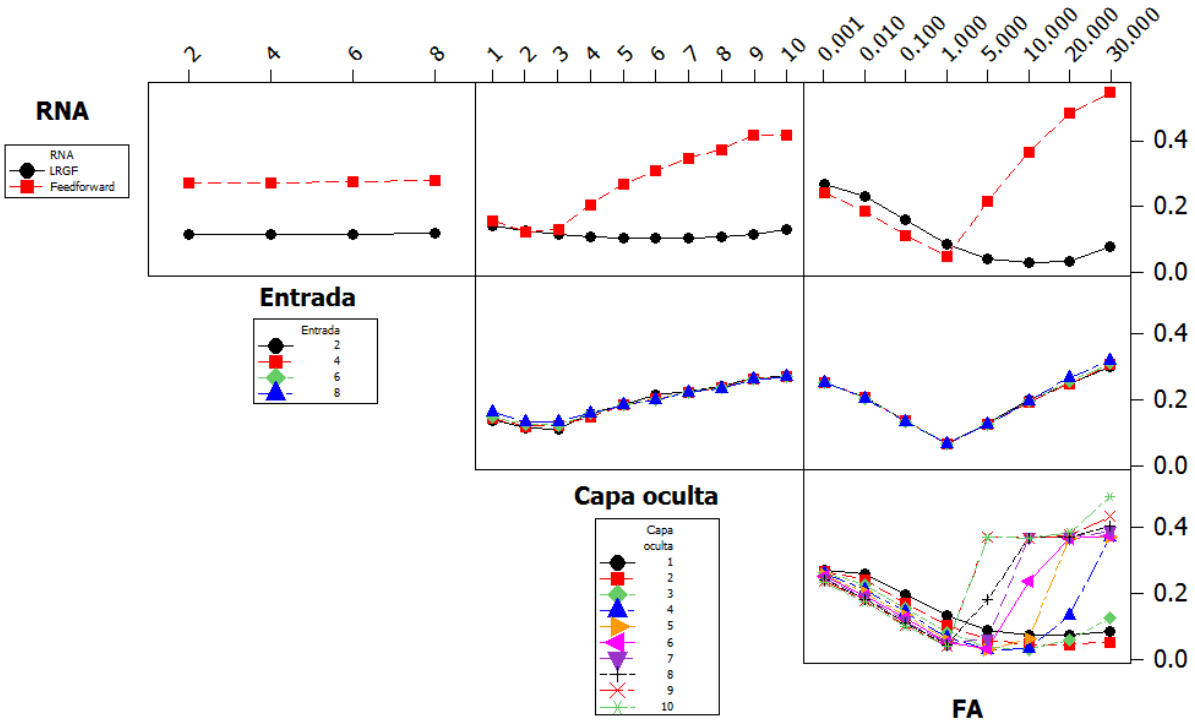


Figura V.2 Gráfica de interacción para EMC.

En este último punto, es donde la gráfica de interacciones separa las medias de los experimentos, dejando más claro el comportamiento individual de ambas redes para el DoE propuesto. Observamos en la Figura V.2 que en la parte superior nos muestra el comportamiento de ambas RNA, en rojo la red “Feedforward” y en negro la red LRGF, por lo que se puede seleccionar la mejor configuración para ambas. Es destacable que ambas redes se comportan de una manera similar, tanto para un numero bajo de

neuronas como para un FA bajo, sin embargo la LRGF muestra mejor comportamiento para la mayoría de las pruebas. De la Figura V.2 se concluye:

- El mejor comportamiento de la red LRGF es con más de tres neuronas, hasta 8 neuronas donde el desempeño empieza a no ser el óptimo.
- Se observa que con una selección mayor a 1 del FA se obtiene resultados satisfactorios.
- El punto que se toma como el mejor por su combinación de número de neuronas y FA es de; siete neuronas en la capa oculta y un FA de cinco. Aunque con un valor más grande del FA se obtiene una ligera mejora en el funcionamiento, estos valores se encuentran cerca del comportamiento errático de la red por lo que se evita seleccionarlos en la configuración óptima.

Con la configuración para la RNA definida, se grafica la simulación de la red LRGF en Matlab (Figura V.3). Esta simulación a servirá de referencia para el diseño en VHDL, obteniendo un un EMC de 0.0394.

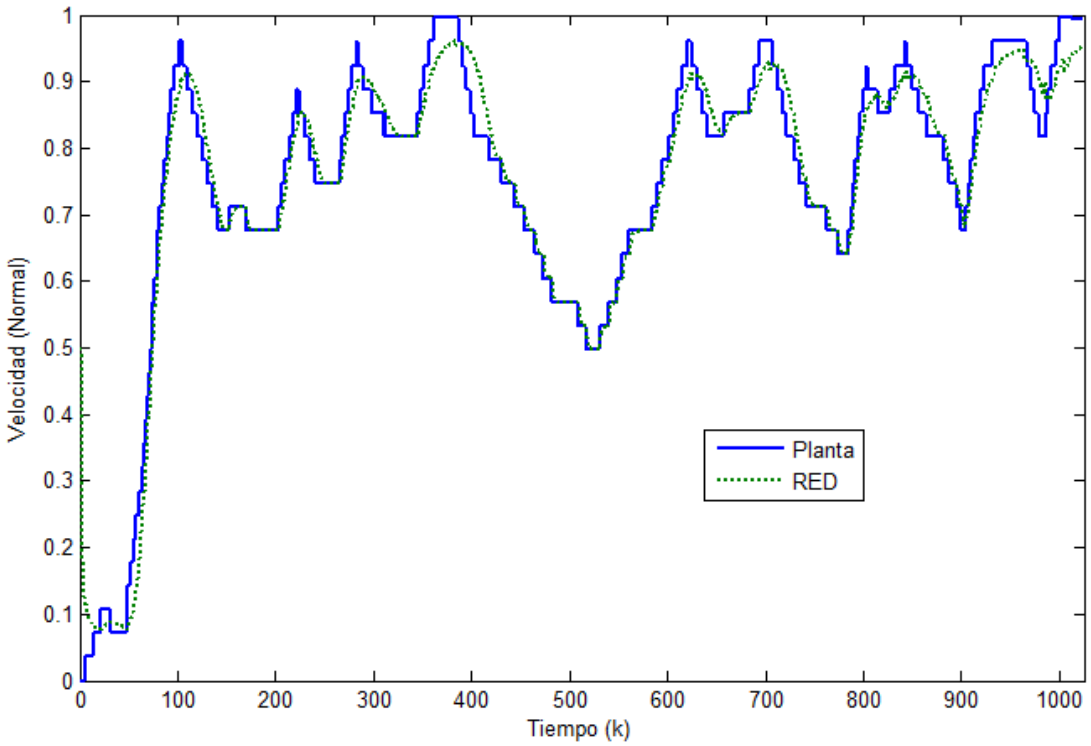


Figura V.3 Simulación en Matlab de la red LRGF con la configuración seleccionada.

Para el diseño final de la red se obtuvo un tiempo ciclo de 660 ns con un reloj principal de operación de 50 Mhz (Figura V.4). Del tiempo de operación la capa oculta ocupa 340 ns en calcular la salida y actualizar los pesos, mientras la capa oculta le toma 220 ns el cálculo de la salida de las neuronas en la capa.

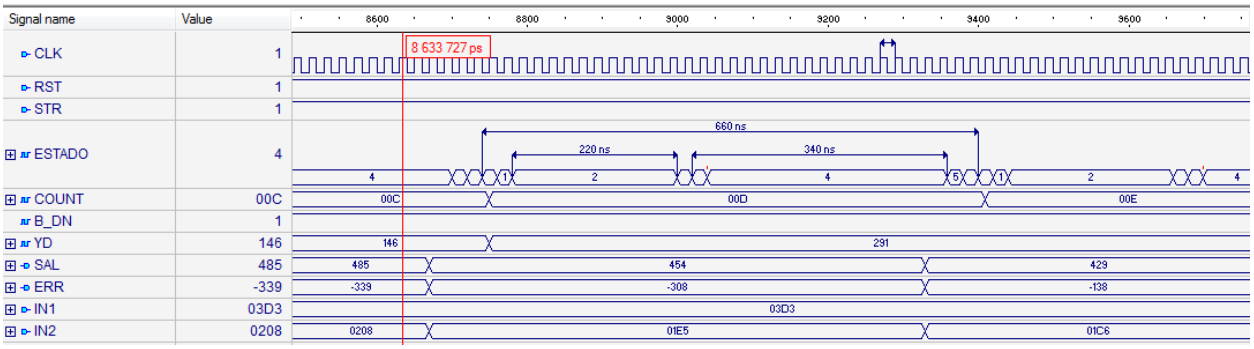


Figura V.4 Diagrama de los tiempos del diseño de la red LRGF en VHDL.

El diseño final de la red fue el mostrado en la Figura IV.23, los resultados obtenidos en el software ActiveHDL para este diseño se pueden observar en la Figura V.5. En este se puede observar los primeros 20 ciclo de operación de la red para los mismos datos usados en Matlab, en estos se puede ver como la red reduce paulatinamente el error de identificación hasta llegar a un -85 en formato 1.12 de punto fijo, que sería un error de -0.0207 en formato decimal.

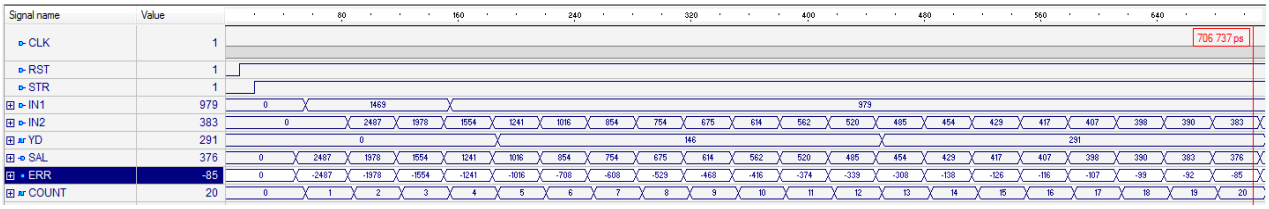


Figura V.5 Resultados del diseño en ActiveHDL.

En la Figura V.6 se muestra la comparación de los resultados obtenidos de la red en Matlab y el diseño en VHDL. Se destaca que aunque dentro del diseño de la RNA en VHDL se restringen los cálculos a un formato de punto fijo, el desempeño es similar al del algoritmo de punto flotante simulado en Matlab. El EMC para el diseño en VHDL es de 0.0539, comparado contra 0.0395 de la simulación en Matlab. Como se puede ver en la Tabla V.2 la velocidad de ejecución del algoritmo en el FPGA fue superior a la observada en la PC, donde el tiempo de ejecución no es constante. La FPGA tarda en calcular todo un ciclo de funcionamiento completo de la RNA LRGF en 660 ns con un oscilador principal de 50 MHz, mientras que el menor tiempo alcanzado dentro de la PC es de solo 226 ms.

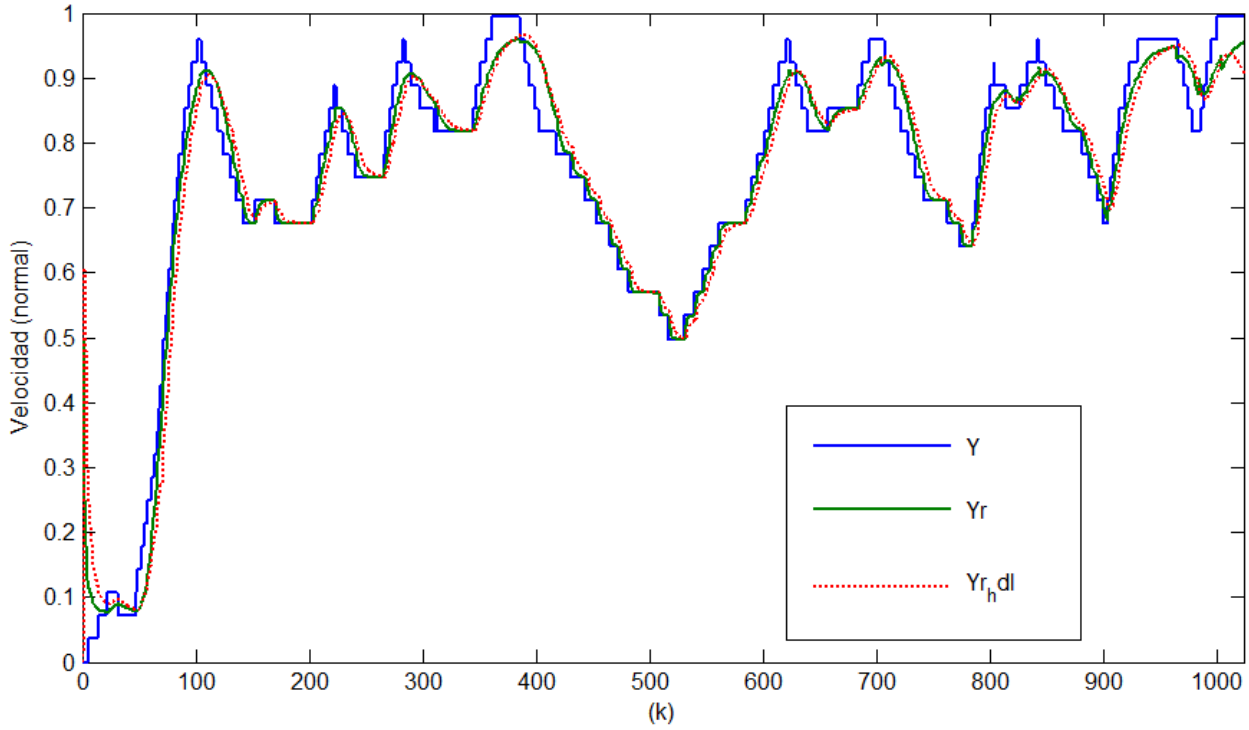


Figura V.6 Comparación de los resultados obtenidos de la simulación en Matlab y el diseño en ActiveHDL.

Tabla V.2 Comparación del desempeño de tiempo entre plataformas.

Plataforma	**Sistema operativo**	**Lenguaje**	**Frecuencia**	**Tiempo de ejecución**
Intel Core i7 2630QM	WINDOWS 7	MATLAB	2.9 GHz	226 ms – 749 ms
spantan-6 XC6SLX45	---	VHDL	50 MHz	660 ns

Se presenta a continuación los resultados de la implementación del diseño de la RNA dentro del FPGA. Para la experimentación se usa el banco de pruebas de la Figura V.7 con el motor de corriente continua Tohuku Ricoh y una tarjeta de desarrollo Atlys (Spartan 6, XC6SLX45) la tabla de utilización de recursos puede verse en el anexo cuatro. Dentro del experimento se usaron los valores ya conocidos del PWM para excitar al motor (Figura V.8).

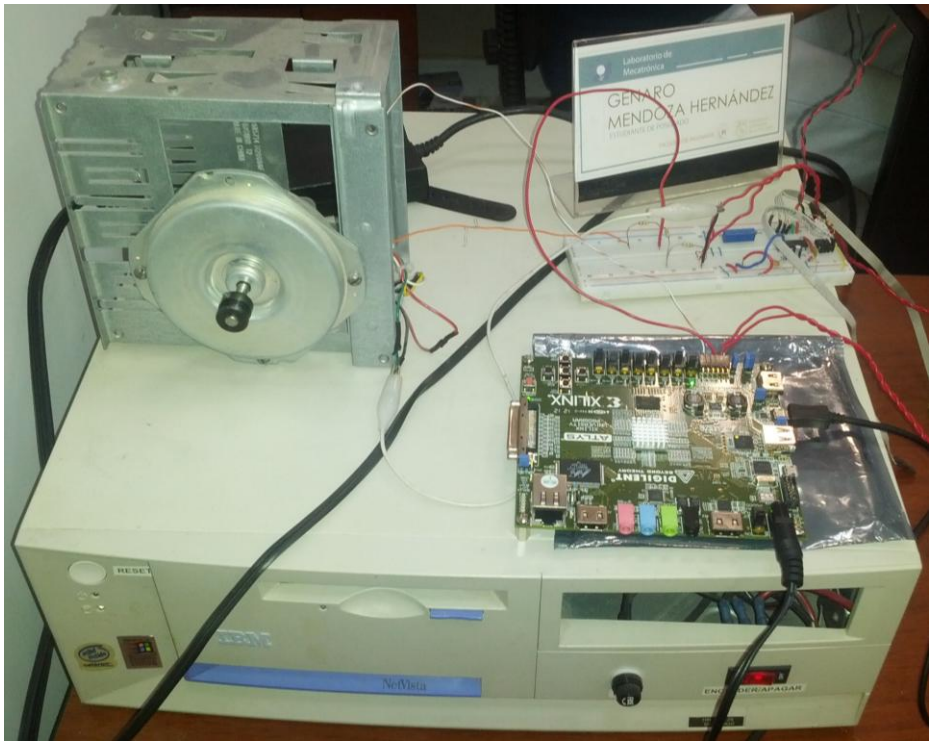


Figura V.7 Banco de pruebas.

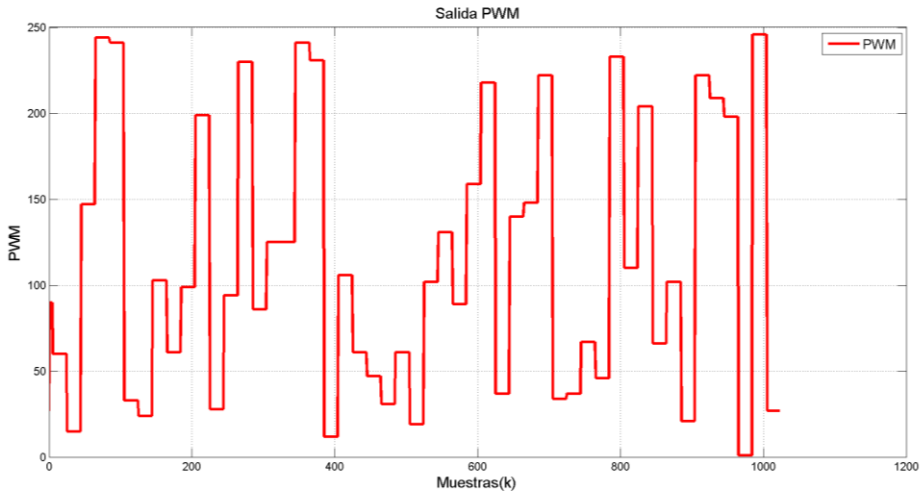


Figura V.8 Señal de PWM utilizada en el experimento.

La respuesta de la red a este Los valores de velocidad obtenidos en este experimento son los aproximados por la FPGA y no son tomados de experimentos anteriores, como los de la Figura V.3. La respuesta “Yr” de la RNA LRGF para la identificación en línea en el FPGA se encuentra en la Figura V.9.

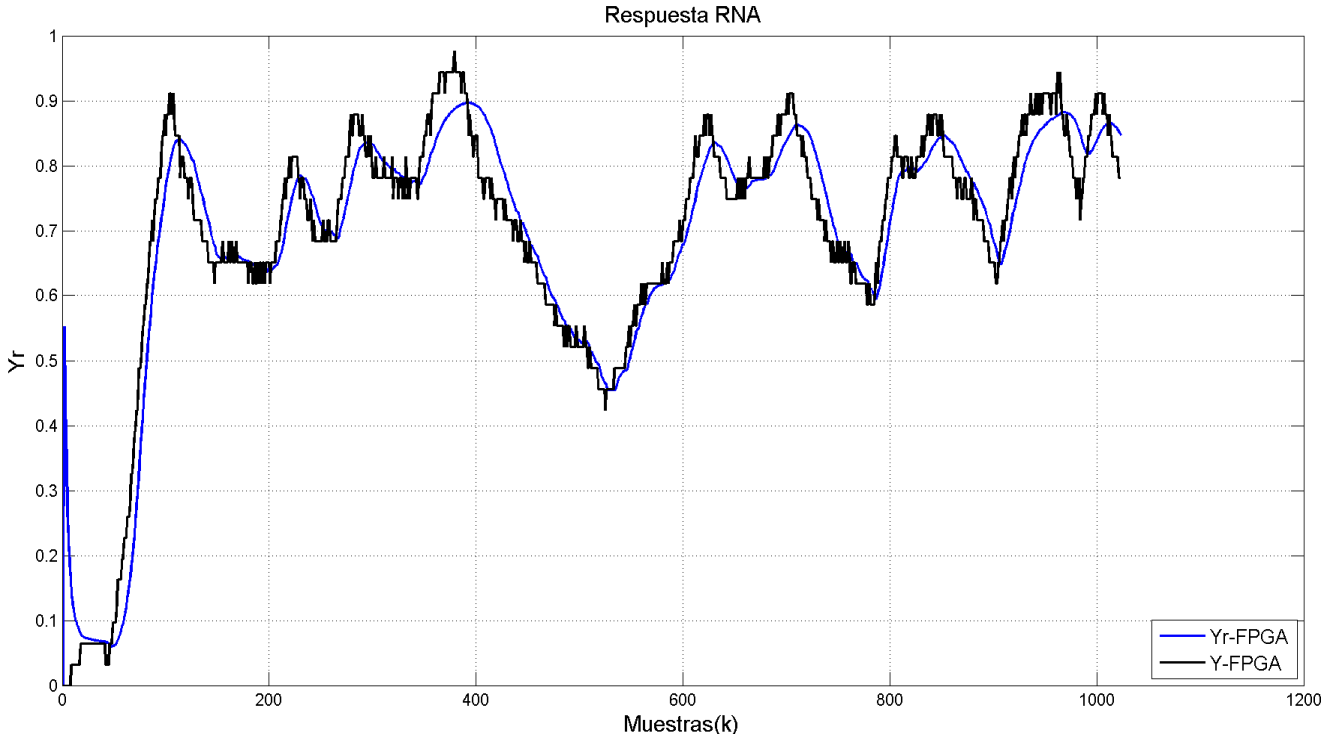


Figura V.9 Identificación en línea del caso de estudio.

Se obtiene un EMC del experimento de 0.0573, resultando en un desempeño similar al alcanzado dentro de la simulación de la RNA en Matlab, donde se resalta que los valores de velocidad son diferentes para cada experimento y no se puede esperar obtener siempre el mismo EMC dentro de los experimentos. Aunque se entiende que ambos experimentos fueron llevados a cabo con diferentes datos de velocidad, se presenta la Figura V.10, para presentar el parecido de la identificación entre plataformas de la RNA LRGF implementada. Se calcula el coeficiente de correlación de Pearson para las respuestas con el resultado de una alta correlación positiva de 0.95.

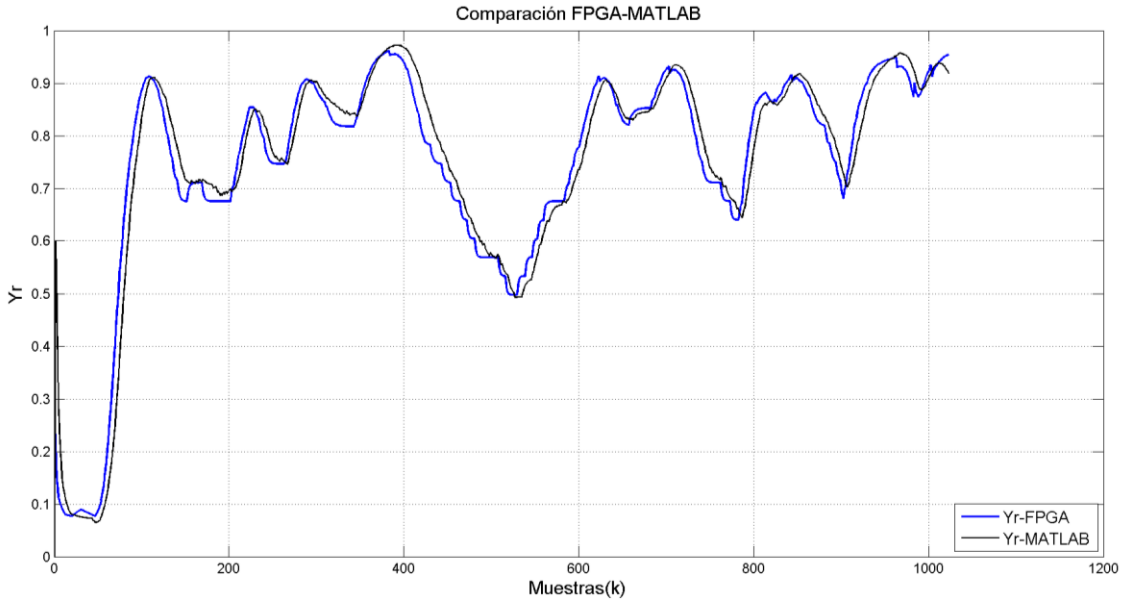


Figura V.10 Respuestas de la RNA en Matlab y el FPGA.

V.2 **CONCLUSIONES**

Se acepta la hipótesis de este trabajo ya que se logró implementar una RNA apta para identificar en tiempo real un sistema no lineal, asimismo se redujo el tiempo de ejecución del algoritmo implementado en FPGA comparándolo contra su símil basado en software.

En la actualidad existen numerosos métodos de convertir código de alto nivel en un diseño en VHDL, sin embargo en este trabajo se diseñó sin la ayuda de estos métodos para tener un mejor control de cómo se calcula el algoritmo dentro de la FPGA. En este trabajo se buscó siempre una paridad entre el número de lógica ocupada y la precisión de la RNA. Queda abierta la posibilidad de incluir esta relación a futuro dentro del Diseño de experimentos para poder tomar mejores decisiones dentro del diseño en VHDL.

Se obtiene buenos resultados en la implementación del diseño de la RNA en FPGA. Siendo el desempeño de la implementación casi idéntico a la contraparte en software. También se aclara que el diseño obtenido de este trabajo está pensado para modificarse tanto en el número de neuronas de entrada y el número de neuronas en la capa oculta, sin embargo esto no es posible con el número de neuronas en la capa de salida.

Los resultados obtenidos del diseño de experimentos ayudan en la elección de parámetros para la configuración de la red en la identificación con una fundamentación estadística, contrario a pruebas que se basan en la apreciación de cada una por separado. Este trabajo es un aporte para generar metodologías que ayuden en la búsqueda de los parámetros más adecuados para cada configuración de red y poder contrastar resultados obtenidos. Queda la posibilidad para trabajos futuros de tomar más factores que afectan la identificación de la red neuronal (función de activación, formato numérico a usar dentro de la implementación), además de diferentes herramientas estadísticas para la búsqueda de los parámetros de una manera más efectiva.

BIBLIOGRAFÍA

Alataris et al., 2000 "A novel network for nonlinear modeling of neural systems with arbitrary point-process inputs". Neural Networks, vol. 13, pág. 255–266.

Ang et al., 2005 “PID control system analysis, design, and technology”. IEEE Transactions on Control Systems Technology 13(4):pp. 559-576.

Carling et al., 1994 "A comparition of 4 methods for nonlinear data modeling". Chemometrics and Intelligent laboratory systems, vol. 23, no. 1, pág. 163-177.

Carlos R. Luna, 2011 “Diseño e Implementación de un algoritmo genético en FPGA para sintonización de controladores PID”. Universidad Autónoma de Querétaro, Querétaro, México.

Chen y Billings, 1992 “Neural networks for nonlinear dynamic systems modeling and identification”. Int. J. of Control, Vol. 56, No 2, 319-346.

Cowan, 1967 “A mathematical theory of central nervous activity”. PhD Thesis, Univ. London, U.K.

Douglas C. Montgomery, 2002 “Diseño y Análisis de Experimentos”. Grupo Editorial Iberoamérica.

Elman, 1988 “Finding Structure in Time”. CRL Technical Report 8801. Center for Research in Language, University of California, San Diego.

Eric Monmasson y Marcian N. Cirstea, 2007 "FPGA Design Methodology for Industrial Control Systems—A Review". IEEE TRANSACTIONS ON INDUSTRIAL ELECTRONICS, VOL. 54, NO. 4, Pag. 1824-1842

Espinosa et al., 2006 “Robust Advanced Pid Control (Rapid), Pid Tuning Based On Engineering Specifications”. IEEE Control Systems Magazine, Pag. 15-19.

Giró et al., 2006 “Evaluación de Modelos Neuronales Destinados a Operar Sistemas Inteligentes de Control”. Mecánica Computacional Vol. 25, Pag. 947-960.

Giró et al., 2007 “Identificación De Parámetros De Sistemas Dinámicos A Través De Redes Neuronales Artificiales”. Mecánica Computacional Vol. 26, Pag. 2585-2599.

Gonzalez y Tang, 2010 “Black-Box Identification of a Class of Nonlinear Systems by a Recurrent Neurofuzzy Network“.IEEE Transactions on neural networks, vol. 21 issue 4.

Haber y Unbehauen, 1990 "Structure identification of non-linear dynamic systems – a survey on input/output approaches". Automatica, vol. 26, pág. 651–677.

Hertz y Palmer, 1991 "An introduction to the theory of neural computation". Lecture notes, vol. 1. Addison-Wesley, Redwood City, CA.

Hopfield, 1982 “Neural Networks and Physical Systems with Emergent Collective Computational Abilities”. Proc. Nat. Acad. Sci., U. S., vol. 79, 2554-2558.

Jordan, 1986 “Serial Order: A Parallel Distributed Processing Approach”.

Kabir et al., 2010 "High-Dimensional Neural-Network Technique and Applications to Microwave Filter Modeling". IEEE Transactions on Microwave Theory and Techniques, vol. 58, issue 1, pág. 145-156.

Karali et al., 2007 "Artificial neural network based prediction of drill flank wear from motor current signals". Applied Soft Computing Volume 7, Issue 3, Pages 929-935.

Kohonen, 1990 “The Self-Organizing Map”. Proc of the IEEE, vol. 78, pág. 1464-1480.

Kung et al., 2009 "FPGA-based self-tuning PID controller using RBF neural network and its application in X-Y table". IEEE International Symposium on Industrial Electronics, 2009. ISIE 2009. Pag. 694-699.

Lin y Lee, 1996 "Neural fuzzy systems : a neuro-fuzzy synergism to intelligent systems". Prentice Hall PTR, Upper Saddle River, N.J. 07458.

Ljung, L. 1999 “System identification – Theory for the User”. Prentice Hall, Upper Saddle River, N.J., 2nd edition.

Martins et al., 2007 "Unsupervised Neural-Network-Based Algorithm for an On-Line Diagnosis of Three-Phase Induction Motor Stator Fault". IEEE Transactions on Industrial Electronics, vol. 54, issue 1, pág. 259-264.

Mastorocostas y Theocharis, 2002 “A Recurrent Fuzzy-Neural Model for Dynamic system Identification”. IEEE Transactions on Systems, Man, and Cybernetics,Part B, Vol. 32, No. 2, pp. 176–190.

McCulloch y Pitts, 1943 “A logical Calculus of the Ideas Immanent in Nervous Activity”. Bull. Math. Biophys, 5, 115-133.

Minsky y Papert, 1969: “Perceptrons”. The MIT Press.

Moody y Darken, 1988 “Learning with localized receptive fields”. Proceedings of the 1988 Connectionist Models Summer School, eds. Touret-zky, Hinton and Sejnowski. Morgan-kaufmann, Publishers.

Nallasivam et al., 2008 “Stiction Identification in Nonlinear Process Control Loops”. American Control Conference, Pag. 1-3.

Narendra y Parthasarathy, 1990 "Identification and control of dynamical systems using neural networks " IEEE Transactions on Neural Networks, Vol. 1 Issue:1 , 4-27.

Norgaard et al., 2003 “Neural Networks For Modelling And Control Of Dynamic System: A Practioner´s Handbook”, Springer.

O´Dwyer, 2006 “Handbook of PI and PID Controller Tuning Rules 2nd Edition” Imprerial College Press.

Olurotimi, 1994 “Recurrent Neural Network Training with Feedforward Complexity”. IEEE Transactions on Neural Networks, Vol. 5, No. 2, pp. 185–197.

Paz, 2006 “Análisis, Síntesis Y Construcción De Un Controlador Adaptable Genérico Con Supervisión Inteligente” CNIDT, departamento de electrónica.

Pham y Xing, 1995 "Neural networks for identification, prediction and control". Springer-Verlag, London, Great Britain.

Polat y Yildirim, 2010 “FPGA implementation of a General Regression Neural Network: An embedded pattern classification system”. Digital Signal Processing Volume 20, Issue 3, Pag. 881-886.

Prokhorov, 2007 “Training Recurrent Neurocontrollers for Real-time Aplication”. IEEE Transactions on neural networks, vol. 18 no. 4.

Rosenblatt, 1958 “The Perceptron: A Probabilistic Model for Information Storage and Organization in the Brain”. Psycholog. Rev., 65, 386-108.

Rumelhart et al., 1986 “Learning Internal Representations by Error Propagation”. Parallel Distributed Processing,MIT Press.

Thomas, 2004 “Control and operations: when does controllability equal profitability?” Computers & Chemical Engineering Volume 29, Issue 1, Pages 41-49.

Tsoi y Back, 1994 “Locally Recurrent Globally Feedforward networks: A Critical Review of Architectures”. IEEE Transactions on Neural Networks, Vol. 5, No. 2, pp. 229–239.

Warwick et al., 1992 "Neural networks for control and systems". Vol. 46 of IEE Control Engineering Series, Peter Peregrinus Ltd., Six Hills Way, Stevenage, London, United Kingdom, 1st Ed.

Widrow y Hoff, 1960 “Adaptive Switching Circuits”. IRE WESCON Conv. Record, Part 4, 96-104.

Yu et al., 2011 "Advantages of Radial Basis Function Networks for Dynamic System Design". IEEE Transactions on Industrial Electronics, vol. 58, issue 12, pág. 5438-5450.

Yuan et al., 2010 "Neural Network Based Self-Learning Control Strategy for Electronic Throttle Valve". Vehicular Technology, IEEE Transactions on, On page(s): 3757 - 3765, Vol. 59 Issue 8.

Yusuf Altintas, 2000 "Manufacturing Automation". Cambridge University Press, The Pit Building, Trumpington Street, Cambridge, United Kingdom.

Zurada, 1992 "Artificial Neural Systems". West Publishing Company. St. Paul, MN.

Anexo 1

```
%%Comparacion de Resultados entre las RNA (Feedforward y LRGF)

clear all;
clc;

NN=10;  %numero de neuronas en la capa oculta
NP=4;   %nuemero de replicas para el exp.
NFA=8;  %nuemero de elementos en el vector FA
NEN=4;  %numero de RETARDOS EN LAS entradas a la red

load Data_DoE

%%Normalizacion de Vel y PWM
for i=1:1024
    for j=1:5
    PWM(i,j)=PWM(i,j)/255;
    aux=max(Vel);
    Vel(i,j)=Vel(i,j)/4500;
    end
end

%%
FA_V=[30 20 10 5 1 .1 .01 .001];

FEEDFORWARD_ident;      %llamado a las funciones de las redes.
RECURRENTE_ident;       %llamado a las funciones de las redes.

%clear Sum_E Sum_ER;
ContornoF_gral = zeros(10,8,4);
ContornoR_gral = zeros(10,8,4);

ContornoF_gral = Sum_E(:,:,1)+Sum_E(:,:,2)+Sum_E(:,:,3)+Sum_E(:,:,4);
ContornoR_gral = Sum_ER(:,:,1)+Sum_ER(:,:,2)+Sum_ER(:,:,3)+Sum_ER(:,:,4);

ContornoF_gral = ContornoF_gral/4;
ContornoR_gral = ContornoR_gral/4;

%for i=1:NEN
figure(NEN);
subplot(1,2,1); surf (ContornoR_gral); title('Pruebas Red
LRGF');XLABEL('FA'); yLABEL('# Neuronas capa oculta'); ZLABEL('Error medio
caudratico');
subplot(1,2,2); surf (ContornoF_gral); title('Pruebas Red
Feedforward');XLABEL('FA'); yLABEL('# Neuronas capa oculta'); ZLABEL('Error
medio cuadratico');
%end;
```

Anexo 2

```
% RED NEURONAL FEEDFORWARD
% numero de neuronas

clc;
clear Sum_E;

for factor=1:NFA  %FA
for main=1:NP
    for test1=1:NN     %%  Banco de prueba
    clear WD WI WO;
    clear WD_BP WI_BP WO_BP;
    WO_max = 0;
    WO_max_g(test1) = 0;
    WO_min = 10;
    WO_min_g(test1) = 10;
    WD_max = 0;
    WD_max_g(test1) = 0;
    WD_min = 10;
    WD_min_g(test1) = 10;
    WI_max = 0;
    WI_max_g(test1) = 0;
    WI_min = 10;
    WI_min_g(test1) = 10;

    n = NEN*2; % # entradas                              i = ...4
    l = test1; % # neuronas en la capa oculta       j = ...3
    m = 1; % # neuronas en la capa de salida        k = ...1

    Tam_m = 1024;
    time = 1:Tam_m;
    FA = FA_V(factor); %%test

    %%  Inicializacion  de pesos

    for i=1:n
        for j=1:l
        WI(i,j)=.00;
            for k=1:m
            WD(j,1)=.00;
            WO(k,j)=.00;
             end
        end
    end

    %% inicio de variables a cero...
    for j=1:l
        H_tm1(j)= 0;
        delta_tm1(j) = 0;
            for i = 1 : n
                DELTA_tm1(i,j) = 0;
            end
    end
```

```
    %% Proceso principal
    for t = 1:Tam_m
        %% planta salida vector X(i)

        Y(k)= Vel(t,main);  % Salida
        U(k)= PWM(t,main); % Entrada

        for i=1:NEN;
            if(t>i);
                X(i)=U(t-i);
                X(i+NEN)=Yr(t-i);
            else

                X(i)=0;
                X(i+NEN)=0;
            end
        end


        %% Calculando salida de la Red

        for j = 1 : l;
            sum = 0;
                for i = 1 : n;
                sum = sum + WI(i,j)*X(i) ;%+ H_tm1(j)*WD(j);
                end
            S(j) = sum + H_tm1(j)*WD(j);
            H(j) = 1/(1 + exp(-S(j)));
            end

            sum = 0;
                for j = 1 : l;
                sum = sum + WO(j)*H(j); %+ V(n+i,j);
                end

            G     = sum;
            Yr(k) = 1/(1 + exp(-G));
            e(k)  = -Y(k)+ Yr(k);

          %% Backpropagation
            for j = 1 : l;

                %for k = 1 : m;
                WO_BP(j)= -e(k)*(Yr(k)*(1-Yr(k)))*H(j);
                %end
                delta(j) = H(j)*(Yr(k)*(1-Yr(k)))*(1-H(j))*(H_tm1(j) +
WD(j)*delta_tm1(j));
                WD_BP(j,1) =  -e(k)*WO(j)*delta(j);

                for i = 1 : n;
                DELTA(i,j) = H(j)*(Yr(k)*(1-Yr(k)))*(1-H(j))*(X(i)+
WD(j)*DELTA_tm1(i,j));
                WI_BP(i,j) = -e(k)*WO(j)*DELTA(i,j);
```

```
            end
        end

        %% Actualizacion de valores
        for j = 1 : l
        WO = WO+WO_BP*FA;
        WI = WI+WI_BP*FA;
        WD = WD;%+WD_BP*FA;

        H_tm1(j) = H(j);
        delta_tm1(j) = delta(j);
            for i = 1 : n
            DELTA_tm1(i,j) = DELTA(i,j);
            end
        end

        WO_max=(max(WO));
            if WO_max > WO_max_g(test1)
                WO_max_g(test1)=WO_max;
            end;
        WO_min=(min(WO));
            if WO_min < WO_min_g(test1)
                WO_min_g(test1)=WO_min;
            end;
        WD_max=(max(WD));
            if WD_max > WD_max_g(test1)
                WD_max_g(test1)=WD_max;
            end;
        WD_min=(min(WD));
            if WD_min < WD_min_g(test1)
                WD_min_g(test1)=WD_min;
            end;
        WI_max=max(max(WI));
            if WI_max > WI_max_g(test1)
                WI_max_g(test1)=WI_max;
            end;
        WI_min=min(min(WI));
            if WI_min < WI_min_g(test1)
                WI_min_g(test1)=WI_min;
            end;
    end

    sum=0;
    for aux=1:1024
        sum = abs(e(aux)*e(aux))+sum;
    end
    Sum_E(test1,factor,main) = sqrt(sum/Tam_m);
    %% SEGUIMIENTO DE LA RED
    % figure(test1)
    % plot(time,Y,'--',time,Yr,'linewidth',1);
    % %axis([0,400,-.3,.3]);
    % xlabel('Tiempo ');
    % ylabel('Amplitud (Rad)');
    % legend('Planta','RED');
    %% ERROR DE APROXIMACION
    % figure(test1+10)
```

```
    % plot (time,e,'red','linewidth',1);
    % %axis([0,400,-.1,.1]);
    % xlabel('Tiempo');
    % ylabel('Amplitud');
    % legend('Error');
    end
%% ERROR DE LA CORRIDA PARA CADA NEURONA
% figure(main)
% plot(1:NN,Sum_E(1:NN,main));
% title('Prueba con dato 1-5 "Red Feedforward"')
% xlabel('Numero de Neuronas en la capa oculta');
% ylabel('Amplitud Error Medio Cuadratico(EMC)');
% legend('EMC');
end
end

% plot(1:10,Sum_E,1:10,Sum_EF)
% xlabel('Numero de Neuronas en la capa oculta');
% ylabel('Amplitud Error Medio Cuadratico(EMC)');
% legend('Recurreeente','Feedforward');
%
% figure(3)
% plot(time,Y,'linewidth',2);
% axis([0,400,-.3,.3]);
% xlabel('Tiempo');
% ylabel('Amplitud');
% legend('Posicion Angular del Pendulo');

%grid on;
%
% figure(2)
% plot (time,Y,time,U,time,e,'red',time,Yr,'green','linewidth',2);
% %axis([0,2000,-.5,.5]);
% legend('"Salida de la Planta"','Entrada a la Planta','Error','RED');
% grid on;
```

Anexo 3

```
% RED NEURONAL RECURRENTE LRGF

clc;
clear Sum_ER;

for factor=1:NFA  %%FA
for main=1:NP
    for test1=1:NN     %%  Banco de prueba
    clear WD WI WO;
    clear WD_BP WI_BP WO_BP;
    WO_max = 0;
    WO_max_g(test1) = 0;
    WO_min = 10;
    WO_min_g(test1) = 10;
    WD_max = 0;
    WD_max_g(test1) = 0;
    WD_min = 10;
    WD_min_g(test1) = 10;
    WI_max = 0;
    WI_max_g(test1) = 0;
    WI_min = 10;
    WI_min_g(test1) = 10;

    n = NEN*2; % # entradas                              i = ...4
    l = test1; % # neuronas en la capa oculta       j = ...3
    m = 1; % # neuronas en la capa de salida         k = ...1

    Tam_m = 1024;
    time = 1:Tam_m;
    FA = FA_V(factor); %%test


    %%  Inicializacion  de pesos

    for i=1:n
        for j=1:l
        WI(i,j)=.00;
            for k=1:m
            WD(j,1)=0;
            WO(k,j)=.00;
             end
        end
    end


    %% inicio de variables recurrentes a cero...
    for j=1:l
        H_tm1(j)= 0;
        delta_tm1(j) = 0;
            for i = 1 : n
                DELTA_tm1(i,j) = 0;
            end
    end
```

```
    %% Proceso principal
    for t = 1:Tam_m
        %% planta salida vector X(i)

        Y(k)= Vel(t,main);  % Salida
        U(k)= PWM(t,main); % Entrada

        for i=1:NEN;
            if(t>i);
                X(i)=U(t-i);
                X(i+NEN)=Yr(t-i);
            else

                X(i)=0;
                X(i+NEN)=0;
            end
        end

             for j = 1 : l;
            sum = 0;
                for i = 1 : n;
                sum = sum + WI(i,j)*X(i) ;%+ H_tm1(j)*WD(j);
                end
            S(j) = sum + H_tm1(j)*WD(j);
            H(j) = 1/(1 + exp(-S(j)));
            end

            sum = 0;
                for j = 1 : l;
                sum = sum + WO(j)*H(j); %+ V(n+i,j);
                end

            G     = sum;
            Yr(k) = 1/(1 + exp(-G));
            e(k)  = -Y(k)+ Yr(k);

          %% Backpropagation
            for j = 1 : l;

                %for k = 1 : m;
                WO_BP(j)= -e(k)*(Yr(k)*(1-Yr(k)))*H(j);
                %end
                delta(j) = H(j)*(Yr(k)*(1-Yr(k)))*(1-H(j))*(H_tm1(j) +
WD(j)*delta_tm1(j));
                WD_BP(j,1) =  -e(k)*WO(j)*delta(j);

                for i = 1 : n;
                DELTA(i,j) = H(j)*(Yr(k)*(1-Yr(k)))*(1-H(j))*(X(i)+
WD(j)*DELTA_tm1(i,j));
                WI_BP(i,j) = -e(k)*WO(j)*DELTA(i,j);
                end
            end
```

```
            %% Actualizacion de valores

            WO = WO+WO_BP*FA;
            WI = WI+WI_BP*FA;
            WD = WD+WD_BP*FA;

            for j = 1 : l
            H_tm1(j) = H(j);
            delta_tm1(j) = delta(j);
                for i = 1 : n
                DELTA_tm1(i,j) = DELTA(i,j);
                end
            end

             WO_max=(max(WO));
                if WO_max > WO_max_g(test1)
                    WO_max_g(test1)=WO_max;
                end;
            WO_min=(min(WO));
                if WO_min < WO_min_g(test1)
                    WO_min_g(test1)=WO_min;
                end;
            WD_max=(max(WD));
                if WD_max > WD_max_g(test1)
                    WD_max_g(test1)=WD_max;
                end;
            WD_min=(min(WD));
                if WD_min < WD_min_g(test1)
                    WD_min_g(test1)=WD_min;
                end;
            WI_max=max(max(WI));
                if WI_max > WI_max_g(test1)
                    WI_max_g(test1)=WI_max;
                end;
            WI_min=min(min(WI));
                if WI_min < WI_min_g(test1)
                    WI_min_g(test1)=WI_min;
                end;
    end

    sum=0;
    for aux=1:1024
        sum = abs(e(aux)*e(aux))+sum;
    end
    Sum_ER(test1,factor,main) = sqrt(sum/Tam_m);
    %% SEGUIMIENTO DE LA RED
%     if (test1 == 8)
%     figure(test1)
%     plot(time,Y,'--',time,Yr,'linewidth',1);
%     %axis([0,400,-.3,.3]);
%     xlabel('Tiempo ');
%     ylabel('Amplitud');
%     legend('Planta','RED');
%     end
    %% ERROR DE APROXIMACION
```

```
%     if (test1 == 8)
%
%     figure(test1+10)
%     plot (time,e,'red','linewidth',1);
%     %axis([0,400,-.1,.1]);
%     xlabel('Tiempo');
%     ylabel('Amplitud');
%     legend('Error');
%     end
end
%% ERROR DE LA CORRIDA PARA CADA NEURONA
% figure(main)
% plot(1:NN,Sum_ER(1:NN,main));
% title('Prueba con dato 1-5 "Red Recurrente"')
% xlabel('Numero de Neuronas en la capa oculta');
% ylabel('Amplitud Error Medio Cuadratico(EMC)');
% legend('EMC');
end
end

% plot(1:10,Sum_E,1:10,Sum_EF)
% xlabel('Numero de Neuronas en la capa oculta');
% ylabel('Amplitud Error Medio Cuadratico(EMC)');
% legend('Recurreeente','Feedforward');
%
% figure(3)
% plot(time,Y,'linewidth',2);
% axis([0,400,-.3,.3]);
% xlabel('Tiempo');
% ylabel('Amplitud');
% legend('Posicion Angular del Pendulo');

%grid on;
%
% figure(2)
% plot (time,Y,time,U,time,e,'red',time,Yr,'green','linewidth',2);
% %axis([0,2000,-.5,.5]);
% legend('"Salida de la Planta"','Entrada a la Planta','Error','RED');
% grid on;
```

Anexo 4

Descripción del número bloques lógicos	Usados	Disponibles	Utilización
Registro	896	54,576	1%
Número de bloques usados como Flip Flops	868		
Número de bloques usados como Latches	28		
LUTs	1,247	27,288	4%
Número de bloques usados como lógica	1,205	27,288	4%
Número de bloques ocupados	438	6,822	6%
Número de LUTs Flip Flop	1,406		
Número con un Flip Flop inutilizado	610	1,406	43%
Número con un LUT inutilizado	159	1,406	11%
Número con un LUTs Flip Flop completamente utilizado	637	1,406	45%
Número de IOBs ligadas	15	218	6%
Número de LOCed IOBs	15	15	100%
Número de RAMB8BWERs	1	232	1%
Número de BUFIO2/BUFIO2_2CLKs	1	32	3%
Número de DSP48A1s	21	58	36%
Número de PLL_ADVs	1	4	25%