Ing. Gerardo Leyva Soto

Implementación de una red neuronal autoajustable en un FPGA para un servomotor.

2008

Universidad Autónoma de Querétaro
Facultad de Ingeniería

Tesis:

Implementación de una red neuronal autoajustable en un FPGA para un servomotor.

Que como parte de los requisitos para obtener el grado de Maestro en Ciencias de la

Ingeniería

Presenta

Ing. Gerardo Leyva Soto

Santiago de Querétaro.

Universidad Autónoma de Querétaro
Facultad de Ingeniería
Maestría en Ciencias con Línea Terminal en
Instrumentación y Control Automático

TESIS:

Implementación de una red neuronal autoajustable en un FPGA para un servomotor.

Que como parte de los requisitos para obtener grado de

Maestro en Ciencias

Presenta:
Ing. Gerardo Leyva Soto.

Dirigido por:
Dr. René de Jesús Romero Troncoso.

SINODALES

Dr. René de Jesús Romero Troncoso
Presidente — Firma

Dr. Pedro Daniel Alaníz Lumbreras
Secretario — Firma

Dr. Rodrigo Castañeda Miranda
Vocal — Firma

Dr. Roque Alfredo Osornio Ríos
Suplente — Firma

M.C. Juan José García Escalante
Suplente — Firma

Dr. Gilberto Herrera Ruíz
Director de la Facultad

Dr. Luis Gerardo Hernández Sandoval
Director de Investigación y Posgrado

Centro Universitario
Querétaro, Qro.
Enero de 2008
México

RESUMEN

En el presente estudio se realizó la implementación del algoritmo de aprendizaje de retropropagación de una Red Neuronal Artificial Autoajustable (RNAA) en un *FPGA (Field Programmable Gate Array, Arreglos lógicos programables en campo*), para el control de lazo cerrado de un servomotor de corriente directa. La metodología del análisis para la descripción del algoritmo de aprendizaje de Retropropagación en el software descriptivo VHDL (*Very high speed integrated circuit Hardware Description Languaje, lenguaje descriptivo de circuitos integrados de muy alta velocidad*), fue la representación digital o de bloques lógicos de cada una de las ecuaciones del algoritmo de aprendizaje de la RNAA, para poder realizar una descripción más fácil de cada una de estas ecuaciones en el software descriptivo VHDL. Terminada la descripción de cada ecuación del algoritmo de aprendizaje, se describió la programación completa de este algoritmo de la RNAA. Se desarrolló la simulación de cada una de ecuaciones del algoritmo de entrenamiento, así como la simulación en conjunto de este en el software descriptivo VHDL, obteniendo resultados satisfactorios de cada una de ellas. La descripción del algoritmo de aprendizaje de la red neuronal autoajustable en el software descriptivo VHDL para la síntesis en el FPGA se realizó con éxito. En el desarrollo de la ingeniería se describió un control inteligente con una red neuronal artificial autoajustable en un solo circuito programable para un servomotor de corriente directa.

Palabras Clave: FPGA. (Arreglos lógicos programables en campo), RNAA.(Red Neuronal Artificial Autoajustable), VHDL.(Lenguaje descriptivo de circuitos integrados de muy alta velocidad)

Summary

In this study, the implementation of a backpropragation learning algorithm of a self-adjusting neural artificial network (RANN) in FPGA (Field Programmable Gate Array) for the closed-loop control of a direct current servo motor was carried out. The methodology of the analysis to describe the backpropragation learning algorithm in the descriptive software VHDL (Very High speed integrated circuit Hardware Description Languaje), included the digital representation of logic units of each one of the equations of the RNAA learning algorithm, in order to make an easier description of each one of them in the descriptive software VHDL. Once the description of each equation in the learning algorithm was finished, the complete programming of the RNAA learning algorithm was described. Simulation of each of the equations in the learning algorithm as well as the simulation for the whole algorithm in the descriptive software VHDL was perfomed, obtaining satisfactory results from each one them. The learning algorithm description in the self-adjusting neural network in the descriptive software VHDL for the synthesis in the FPGA was performed successfully. In the development of engineering, an intelligent control with a self-adjusting neural artificial network in a single programmable circuit for a direct current servo-motor was described.

Key Words: FPGA. (Field Programmable Gate Array), RNAA.(self-adjusting neural artificial network), VHDL. (Very High speed integrated circuit Hardware Description Languaje)

DEDICATORIAS

En particular a la mujer que me apoyo y me tuvo paciencia para poder alcanzar esta meta en mi vida profesional la de obtener el grado de maestro en ciencias de la ingeniería, por que sin ti no hubiera logrado todo esto, gracias Elizabeth te amo.

Gracias

A mis hijos por su paciencia al no estar todo el tiempo con ellos

Gracias

AGRADECIMIENTOS

Al Doctor René de Jesús Romero Troncoso por su paciencia, su buena dirección, y por su profesionalismo mostrado para el desarrollo de esta tesis.

Gracias.

Al Doctor Gilberto Herrera por todo su apoyo en mi estancia en la maestría.

Gracias

A Carlos Mireles por sus comentarios oportunos, y a toda la asesoría y ayuda en toda la maestría.

Gracias

Al CBTis 118 por todo su apoyo brindado durante mis estudios de posgrado.

Gracias

Í N D I C E

CAPÍTULO II

CAPÍTULO III

CAPÍTULO IV

INDICE DE CUADROS

INDICE DE FIGURAS

Figura **Página**

Introducción.

Las redes neuronales artificiales basan su funcionamiento en las redes neuronales reales, están formadas por un conjunto de unidades de procesamiento conectadas entre sí. Por analogía con el cerebro humano se denomina "neurona" a cada una de estas unidades de procesamiento. Cada neurona recibe muchas señales de entrada y envía una única señal de salida (como ocurre en las neuronas reales).

Las redes neuronales son una estructura bien definida, y el algoritmo muy apropiado para modelar tipo de sistema. Por lo tanto una red neuronal es un acceso para modelar una planta cualquiera que contenga no linealidades, permitiendo que los parámetros sean variados, también permite un rango grande no linealidades.

Objetivo

El objetivo de la tesis es desarrollar e implementar en un FPGA un sistema de control inteligente modelado por medio de una red neuronal artificial autoajustable para un servomotor de corriente directa.

Hipótesis

Las redes neuronales son un área de la inteligencia artificial que ha evolucionado en los últimos años, por lo que ha despertado mucho interés, en esta área. Se encontró que las redes neuronales son capaces de resolver problemas de control muy complejos, cuya solución por otros métodos convencionales resulta extremadamente difícil. La característica más importante de las redes neuronales es que son capaces de aprender.

Por lo cual en lugar de programar la red, se le enseña una serie de ejemplos, donde las redes neuronales aprenden las relaciones principales que están implícitas en la base de los datos de entrenamiento. Estas relaciones pueden ser no lineales, las redes pueden ser una herramienta muy eficaz y potente para modelar sistemas de procesos industriales de cualquier tipo de complejidad.

Las tecnologías de hoy nos permite aplicar la teoría de redes neuronales y sus métodos de aprendizaje, para esta tesis se utiliza el software de descripción de circuitos integrados de muy alta velocidad VHDL para la descripción del algoritmo de aprendizaje de **propagación hacia atrás o retropropagación**, para que se implemente en un solo circuito programable en un solo circuito programable FPGA para controlar un servomotor de corriente directa en un lazo de control.

Justificación

El desarrollo de riqueza dependen, en gran una gran medida, de cómo son aprovechados los avances de la tecnología. En particular la automatización y la robótica juegan un papel fundamental en la industria de los países desarrollados. El hombre se a caracterizado siempre por la búsqueda constante de nuevas vías para mejorar sus condiciones de vida. Estos esfuerzos les han servido para reducir el trabajo en aquellas operaciones en las que la fuerza juega un papel primordial. Los procesos obtenidos han permitido dirigir estos esfuerzos a otros campos.

El desarrollo de maquinas electrónicas ha permitido implementar fácilmente algoritmos para resolver infinidad de problemas que antes resultaban muy difíciles de resolver. Sin embargo, observamos una limitación importante: ¿Qué ocurre cuando el problema que se requiere resolver no admite un algoritmo tan difícil de realizar, como es el caso, por ejemplo, de la clasificación de objetos por rasgos comunes?

Este ejemplo demuestra que la construcción de nuevas maquinas versátiles que requieren de un enfoque del problema desde otro punto de vista. Así, la inteligencia artificial es un mismo intento por descubrir y describir aspectos de la inteligencia humana que pueden ser simulados mediante máquinas.

Esta disciplina se ha desarrollado fuertemente en los últimos años teniendo aplicaciones en algunos otros campos como la visión artificial, demostración de los teoremas, procesamiento de información expresada mediante lenguajes humanos, controles automáticos inteligentes, etc.

Las redes neuronales artificiales son una opción muy poderosa para desarrollar controles inteligentes que tomen sus propias decisiones, esto ocurre por medio de su entrenamiento o algoritmo de aprendizaje. Los cuales son programados hoy en las computadoras, de acuerdo a los avances tecnológicos estos algoritmos de aprendizaje de redes neuronales pueden ser descritos en otra forma de señales en hardware, logrando así un costo bajo para implementar aplicaciones donde se utilicen Redes Neuronales Artificiales. Así, parece claro que una forma de aproximarse al problema consista en la construcción de sistemas que sean capaces de reproducir esta característica humana.

En definitiva, las redes neuronales son un modelo artificial y simplificado del cerebro humano, que es el ejemplo más perfecto del que disponemos, un sistema que es capaz de adquirir conocimiento a través de la experiencia. Una red neuronal es "un nuevo sistema para el procesamiento de la información cuya unidad básica de procesamiento esta inspirada en la célula fundamental del sistema nervioso humano la neurona".

Descripción del problema

Planteamiento

El problema que se plantea, es analizar los diferentes métodos del software de descripción de circuitos integrados de muy alta velocidad VHDL para la descripción de pasos del algoritmo de Retropropagación de la red neuronal Autoajustable propuesta para implementarla en un FPGA, para el control de un servomotor de corriente directa.

El análisis proporcionará, en primera instancia las bases para la investigaciones subsecuentes relacionas con los problema inherentes con la utilización de redes neuronales artificiales para el control de posición y velocidad de un servomotor de corriente directa.

CAPÍTULO I

REDES NEURONALES

En este capítulo se describen los fundamentos de las redes neuronales. Se comienza dando una breve introducción histórica de su nacimiento y las características más importantes que han hecho de las mismas una herramienta fundamental en la resolución de un gran número de problemas. En cuanto a la clasificación de las redes neuronales podríamos establecer una división de las mismas en función de su arquitectura y forma de aprendizaje de procesar las señales en tres clases.

1.1 Introducción.

En 1949 Hebb propuso una ley de aprendizaje que puede considerarse como el punto de partida de los algoritmos de entrenamiento de las redes neuronales, mientras que la primera estructura de red neuronal fue propuesta por Frank Rosenblatt en 1958. Desde entonces el interés suscitado por las redes neuronales ha crecido enormemente, proponiéndose nuevos modelos que han venido a superar las limitaciones de las redes anteriores, dándoles mayores capacidades. Las redes neuronales tienen su origen en la observación de los sistemas neuronales de los organismos vivientes, sin embargo, éstas no son un fiel reflejo de dichas redes biológicas. Como características más relevantes de las redes neuronales artificiales citamos las siguientes:

1- La capacidad de aprendizaje. La naturaleza no lineal de las mismas les permite exhibir comportamientos verdaderamente complejos. Esta característica junto con algoritmos que extraen un mayor rendimiento de tales sistemas les confiere esa capacidad.

2- La posibilidad de procesamiento paralelo, aspecto que es aprovechado en la implementación, lleva a aumentar considerablemente la velocidad de procesamiento.

3- La significativa tolerancia a fallos, dado que un fallo en unas pocas conexiones locales, rara vez contribuye significativamente al funcionamiento global de la red.

Esta propiedad las hace convenientes en ciertas aplicaciones, donde, una avería puede constituir un serio peligro. Estas características, entre otras, hacen de las redes neuronales una herramienta fundamental en varios campos del conocimiento.
Las redes neuronales generalmente han sido implementadas mediante la programación sobre computadoras digitales, sin embargo ya en nuestros días empiezan a proliferar implementaciones en hardware, ya sea puramente mediante montajes electrónicos, como circuito integrados programables *FPGA* o incorporando dispositivos ópticos. Tales ingenios, presentan un paso importante hacia la consecución de mayores velocidades de procesamiento, así como un camino hacia la popularización de las mismas, siendo ésta un área de investigación muy activa.

En nuestros días la ingeniería de control se enfrenta a sistemas cada vez más complejos, a requisitos de diseño más exigentes y al desconocimiento de las plantas y sus entornos, revitalizando este hecho la investigación en el campo de las redes neuronales.
En cuanto a la clasificación de las redes neuronales podríamos establecer una división de las mismas en función de su arquitectura y forma de procesar las señales en tres clases.
La primera clase de redes se conoce con el nombre de redes neuronales estáticas. En esta clase podemos encontrar la red Multicapa de Perceptrones, la red de Funciones de Base Radial, Red Neuronal Probabilística (*RNP*), etc. Todas ellas tienen como característica común el no poseer memoria, es decir, sólo son capaces

de transformar un conjunto de entradas en un conjunto de salidas, de tal manera que una vez establecidos todos los parámetros de la red las salidas únicamente dependen de las entradas. En general la relación deseada de entradas y salidas se determina en este caso externamente mediante alguna forma de ajuste de los parámetros del sistema supervisado. Este tipo de redes se han empleado con éxito en muchos problemas de clasificación, como funciones lógicas, así como en el campo de la aproximación funcional.

Como segunda clase de redes neuronales debemos nombrar las redes neuronales dinámicas. Estas a diferencia de las anteriores permiten establecer una relación entre salidas y entradas y/o salidas y entradas previas. Esto añade cierta memoria a estas redes. Matemáticamente, esta memoria se traduce en la aparición de ecuaciones diferenciales o ecuaciones en diferencia formando parte del modelo de las mismas.
Como ejemplos de este tipo de redes encontramos la red de Hopfield, la red de retardos en el tiempo (*Time Delay Neural Network*), la red de tiempo discreto (*Time Discrete Neural Network*), etc. Las redes neuronales dinámicas se han revelado útiles en problemas de modelización de la dinámica directa e inversa de sistemas complejos, tales como robots, cohetes, naves espaciales, etc., así como en la modelización de circuitos secuenciales y en la conversión de texto a voz.

Por último podemos destacar las denominadas redes auto-organizativas, en las cuales los nodos vecinos dentro de la red neuronal compiten en actividad por medio de las interacciones mutuas laterales, y evolucionan adaptativamente hacia detectores específicos de las diferentes señales de entrada. El aprendizaje en este caso se denomina aprendizaje competitivo, no supervisado o autoorganizativo. Estas se han aplicado con éxito en problemas de reconocimiento de patrones, control de procesos e incluso en el procesamiento de información semántica.
En las secciones siguientes comentaremos en más detalle cada una de las clases de redes nombradas en los párrafos anteriores.

1.2 Redes neuronales estáticas.

La primera clase de redes se conoce con el nombre de redes neuronales estáticas. La primera red estática es el modelo de neurona más empleado para éstas. En esencia, responde al modelo de neurona introducido por Rosenblatt al que denominó perceptron (Rosenblatt, 1958). Dicha neurona se ilustra en la figura 1.1.

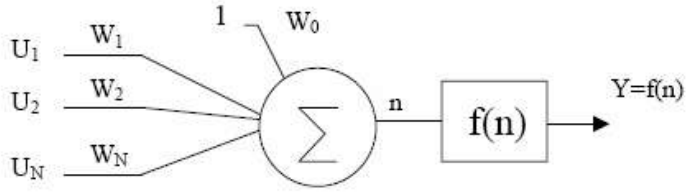


Figura 1.1 – Neurona del Perceptron

Como se puede ver en la Figura (1.1) las entradas se multiplican por una serie de pesos, siendo la suma de esos productos la variable independiente de una función f(x). Dicha función recibe el nombre de función de activación, en analogía con las neuronas biológicas que inspiraron su creación. Además suele ser una función no lineal. En el modelo propuesto originalmente se tomó la función salto unidad.

$$f(n) = \begin{cases} 1 & n > 0 \\ 0 & n \leq 0 \end{cases} \tag{1.1}$$

El resultado de la aplicación de esa función a la suma de productos nos dará la salida de la neurona. Se suele adicionar un peso más que no es multiplicado por ninguna entrada y sirve como una entrada de referencia. Este nuevo peso contribuye generalmente a hacer el aprendizaje más rápido.

Una neurona de este estilo, por tanto, vendrá representada por las ecuaciones que se muestran a continuación:

$$n = \sum_{i}^{N} w_i u_i + w_0 \tag{1.2}$$

$$Y = f(n) \tag{1.3}$$

Donde:

N = número de entradas.

u_i = entrada i-ésima.

w_i = peso i-ésimo.

w_0 = entrada de referencia (peso bias)

$f(n)$ = función de activación

Y = Salida de la neurona.

A pesar de la simplicidad del perceptron, éste es capaz de simular diferentes funciones lógicas, aun cuando el número de las mismas está limitado, denominándose a este límite, umbral de funciones lógicas. También se ha empleado en problemas de clasificación de patrones dentro de dos clases en donde la limitación fundamental se encuentra en que la frontera entre las dos clases puede ser únicamente un hiperplano.

Existen varios algoritmos de entrenamiento diseñados para el perceptron. Entre ellos podemos citar el algoritmo de los mínimos cuadrados, que tiene la peculiaridad de ser un caso particular del algoritmo backpropagation, algoritmo propuesto por Werbos en 1974.Tal como hemos observado el perceptron por si sólo tiene ciertas limitaciones, varios investigadores realizaron experimentos con una capa de perceptrones, al objeto de conseguir aprender ciertas funciones que un único perceptron no permitía aprender. El resultado de tales experimentos fue publicado por Marvin Minsky y Parpert en 1969 en su libro titulado "Perceptrons" donde ponían de manifiesto la incapacidad de una capa de perceptrones para aprender ciertas funciones, incluyendo la función lógica or-exclusiva. Este resultado abrió un período de pesimismo frente a las posibilidades de las redes neuronales estáticas. De hecho tales resultados llevaron al abandono de las redes neuronales durante dos décadas. Sin embargo gracias al esfuerzo de unos pocos investigadores tales como Teuvo Kohonen, Stephen Grossberg y James Anderson las redes neuronales volvieron a coger auge, dado que demostraron que utilizando

varias capas de neuronas se podían solucionar problemas que con una capa eran irresolubles tales como el aprendizaje de una función or-exclusiva. A partir de entonces han surgido diferentes topologías de redes neuronales.

La topología más general de red neuronal, es aquella en donde un conjunto de neuronas se agrupan entre sí. Las salidas de unas neuronas hacen la función de entradas de otras, siendo algunas de las salidas escogidas como salidas de la red neuronal. Un cierto subconjunto de neuronas recibirán las entradas a la red. Este tipo de red neuronal recibe el nombre de red neuronal completamente conexionada, dado que contempla todas las posibilidades de uniones entre neuronas. Un esquema de dicha topología se puede observar en la siguiente figura.

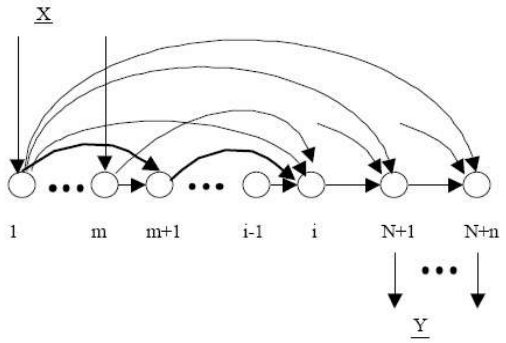


Figura 1.2 – Topología general de una red neuronal.

Donde:

m = número de entradas.

n = número de salidas.

N = indica el número de neuronas existentes sin contar las que dan las salidas de la red.

X = vector de entradas a la red.

Y = vector de salidas de la red.

Sin embargo la mayoría de investigadores prefieren una organización de la red en forma de capas, es decir las neuronas se posicionan en forma de capas unas detrás

de otras, tal que las salidas de las neuronas de unas capas sean las entradas de las neuronas de la capa siguiente. A este tipo de redes en donde las neuronas se disponen en capas sin realimentaciones entre ellas se les da el nombre de redes neuronales feedforward. En estas redes estructuradas por capas se suele distinguir entre la capa de entrada, o sea la capa de neuronas que recibe las entradas a la red, la capa de salida, es decir la capa que contiene las neuronas cuyas salidas serán las que constituyan las salidas de la red y las capas ocultas que son las capas existentes entre las dos anteriores. En la figura 1.3 se muestra una red de este tipo constituida por tres capas.

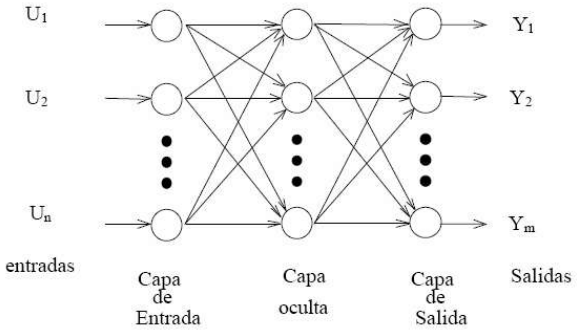


Figura 1.3 – Red neuronal de 3 capas.

Este es el caso de la red Multicapa de Perceptrones (MLP), en donde el modelo de neurona adoptado es el perceptron, ilustrado en los párrafos anteriores. Sin embargo en este caso la función de activación deja de ser la función salto unidad para convertirse en la función sigmoide, la tangente hiperbólica o cualquier otra función no lineal que sea continua, suave y monótonamente creciente. La función salto unidad presenta el inconveniente de no ser derivable, circunstancia ésta, que hace más difícil el diseño de algoritmos de entrenamiento basados en el gradiente.
En ocasiones se han utilizado funciones de activación lineales para las neuronas de la capa de salida, motivado por el hecho de facilitar el aprendizaje. En general se han aplicado varios algoritmos de entrenamiento a la MLP, siendo el más usado el "backpropagation".

La MLP conjuntamente con el "backpropagation" se han empleado con éxito en ciertos problemas de clasificación, pudiendo generar fronteras entre las diferentes clases mucho más complejas que en el caso del perceptron. Investigadores tales como R. P. Lippman en 1987, G. Cybenko en 1989 o Chester en 1990 demostraron de forma teórica que una red MLP de dos capas es capaz de aproximarse a cualquier función no lineal continua, así como implementar cualquier frontera no lineal entre varias clases. Estas características han permitido introducir las redes neuronales en el mundo del control. Citamos a título de ejemplo las aplicaciones a problemas de optimización, en donde se ha revelado como estrategia alternativa a la programación dinámica o la posibilidad de implementar controladores basados en redes neuronales.

Sin embargo el "backpropagation" es un algoritmo inherentemente recursivo, propiedad que se manifiesta de forma negativa en la velocidad de procesamiento del mismo, aunque hemos de destacar que el hecho de ser un algoritmo susceptible de cierta paralelización contribuye a contrarrestar este efecto. Una de las dificultades más significativas de la aplicación de las redes neuronales es la no existencia de métodos para elegir el tamaño óptimo de la red según la aplicación, en general se deja en manos de la experiencia.

Con el objeto de salvar esta dificultad ciertos algoritmos de entrenamiento toman en consideración el variar el tamaño de la red hasta obtener un resultado óptimo. Una característica que condiciona también el aprendizaje es la irregularidad de la superficie de error. Esto contribuye, por un lado a confundir puntos sobre la superficie de error como mínimos globales, mientras que por otro aparecen saltos abruptos que limitan considerablemente la velocidad de aprendizaje, dado que las velocidades de aprendizaje excesivas llevan a inestabilizar el algoritmo. Con el fin de resolver este problema los investigadores introducen términos adicionales en las fórmulas de corrección de los pesos, de tal forma que las actualizaciones sobre los pesos dependan de las actualizaciones previas. De igual forma se intenta evitar el problema de que el algoritmo de aprendizaje quede atrapado en un mínimo local

por medio de métodos especiales de inicialización de los pesos (Wessels *et al*,. 1992; Drago, 1992), mejorando considerablemente los resultados.

En esta clase de redes neuronales podemos encontrar la red Multicapa de Perceptrones, la red de Funciones de Base Radial, Red Neuronal Probabilística (*PNN*), etc. Todas ellas tienen como característica común el no poseer memoria, es decir, sólo son capaces de transformar un conjunto de entradas en un conjunto de salidas, de tal manera que una vez establecidos todos los parámetros de la red las salidas únicamente dependen de las entradas. En general la relación deseada de entradas y salidas se determina en este caso externamente mediante alguna forma de ajuste de los parámetros del sistema supervisado. Este tipo de redes se han empleado con éxito en muchos problemas de clasificación, como funciones lógicas, así como en el campo de la aproximación funcional.

1.3 Redes neuronales dinámicas.

La segunda clases de redes neuronales artificiales son las dinámicas. Estas a diferencia de las anteriores permiten establecer una relación entre salidas y entradas y/o salidas y entradas previas. Esto añade cierta memoria a estas redes. Matemáticamente, esta memoria se traduce en la aparición de ecuaciones diferenciales o ecuaciones en diferencia formando parte del modelo de las mismas.

Las redes neuronales dinámicas se caracterizan por tener una dependencia de sucesos ocurridos en instantes pasados (Nerrand *et al*., 1993). Desde un punto de vista matemático esto se refleja en el hecho de que en las ecuaciones que las definen aparecen ecuaciones diferenciales o ecuaciones en diferencia según sean redes continuas o discretas. Son éstas las que le confieren una cierta memoria, que resulta imprescindible en problemas tales como la identificación de sistemas

dinámicos. La principal atracción de este tipo de redes es que tienen memoria interna y por lo tanto son en sí mismas un sistema dinámico.

Como primer intento de incorporar un comportamiento dinámico en las redes neuronales se diseñaron redes neuronales dinámicas constituidas por una combinación entre redes neuronales estáticas y sistemas dinámicos. En la figura 1.4 representamos cuatro configuraciones conectadas en cascada y con realimentaciones que representan las unidades básicas con las cuales se construye cualquier red dinámica de este tipo (Narendra, 1991). Obsérvese que W(z) hace referencia a la matriz de transferencia del sistema dinámico y N es la red neuronal estática o dado que podemos ver las redes neuronales estáticas como operadores no lineales, también se puede considerar como un operador no lineal.

Cuando en las configuraciones de la figura 1.4 utilizamos varias redes neuronales estáticas denotamos cada una de ellas mediante un superíndice. Mediante estas configuraciones, considerando también el operador multiplicación por una constante, podemos construir una gran variedad de redes dinámicas discretas. Hemos de comentar que las redes consideradas son discretas, sin embargo el análisis se puede extender a las redes dinámicas continuas sin más que tomar matrices de transferencia en el dominio s (W(s)) correspondientes a sistemas dinámicos continuos. Lo que se traduce en el dominio temporal en sustituir los operadores retardo por integradores.

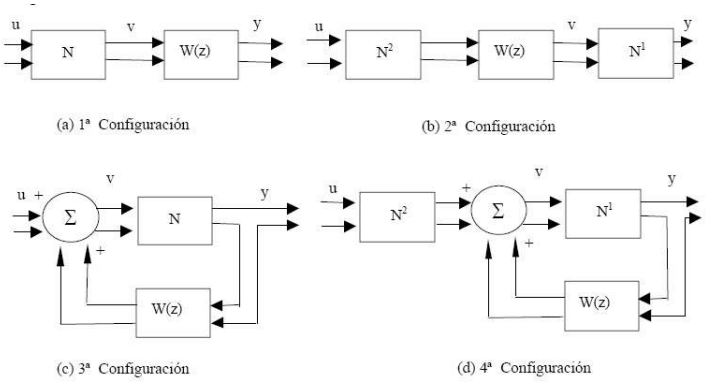


Figura 1.4 – Configuraciones básicas de redes neuronales dinámicas.

Por lo tanto una red neuronal es un acceso para modelar una planta cualesquiera que contenga no linealidades, permitiendo que los parámetros sean variados, también permite un rango grande de no linealidades. El aprendizaje y el entrenamiento de una red neuronal es un procedimiento que representa una planta dinámica, como se muestra en la figura 1.5

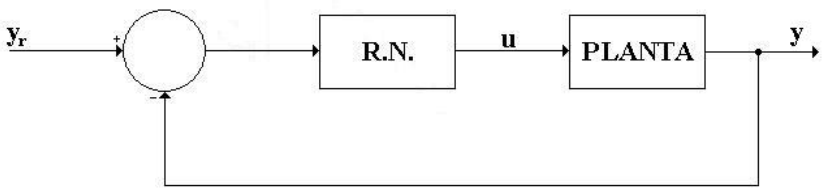


Figura 1.5- Diagrama a bloques de un control de una planta con redes neuronales.

De la figura 1.5, la red neuronal esta en paralelo con la planta y el error, y entre la salida del sistema, y la salida de la red neuronal, la predicción del error, es usado en la señal de enseñanza. Una red neuronal contiene un potencial muy grande para sistemas de control inteligente, porqué aprende y se adapta a los parámetros deseados, con éstas podemos hacer aproximaciones de funciones no lineales, éstas son un de datos de un proceso distribuido en paralelo, y son un modelo natural para los sistemas multivariables.

Como ejemplos de este tipo de redes encontramos la red de Hopfield, la red de retardos en el tiempo (*Time Delay Neural Network*), la red de tiempo discreto (*Time Discrete Neural Network)*, etc. Las redes neuronales dinámicas se han revelado útiles en problemas de modelización de la dinámica directa e inversa de sistemas complejos, tales como robots, cohetes, naves espaciales, etc., así como en la modelización de circuitos secuenciales y en la conversión de texto a voz.

1.4 Redes auto-organizativas.

Las redes auto-organizativas se entienden como una alternativa a las arquitecturas de redes neuronales presentadas en los apartados anteriores. En 1980 Shun-Ichi Amari publicó un trabajo que sentaba las bases del aprendizaje de este tipo de redes, siendo propiamente desarrollado el aprendizaje auto-organizativo a principios de 1981 por Teuvo Kohonen. En este tipo de redes los nodos se ajustan específicamente a varios conjuntos de señales o clases de patrones mediante un proceso de aprendizaje no supervisado. En la versión más básica, solamente un nodo o un grupo local de ellos responden al mismo tiempo a la entrada actual. La localización de los nodos dentro de la red se corresponde con un dominio particular del espacio de entradas. Los nodos se van ordenando como si hubiese un sistema de coordenadas para las diferentes características de las entradas a lo largo de la red. Cada nodo o conjunto de los mismos actúa como un decodificador individual frente a las entradas. Aunque las redes auto-organizativas se han desarrollado más desde un punto de vista de la ingeniería que desde un estudio puramente neurológico, las representaciones internas de la información en el cerebro están generalmente organizadas espacialmente. Así se ha observado que determinadas funciones se encuentran localizadas en determinadas áreas del cerebro. Esta característica es común a las redes de aprendizaje auto-organizativo, siendo gobernadas éstas últimas por principios similares a los que rigen el cerebro.

Esta apreciación justifica el hecho de tomarlas en este capítulo como redes neuronales artificiales. El algoritmo más comúnmente utilizado en estas redes es el algoritmo de aprendizaje auto-organizativo de Kohonen. En resumen éste se basa en la concentración espacial de la actividad de la red sobre los nodos que mejor se ajustan a la entrada actual y en el ajuste adicional del nodo más cercano y sus vecinos a la entrada actual. Este algoritmo ha sido aplicado con éxito en problemas de clasificación de patrones, en diferentes problemas del campo de la robótica, en

el control de procesos, etc. En el caso de los problemas de clasificación, con objeto de aumentar la precisión a la hora de clasificar se han empleado adicionalmente otros algoritmos basados en aprendizaje supervisado como el LVQ (Aprendizaje por Vector de Cuantización, *learning vector quantization*) (Linde, 1980; Gray, 1984; Kohonen, 1990).

Antes de concluir este apartado sería necesario especificar que aun cuando este tipo de redes es diferente de los presentados en las secciones anteriores carece al igual que en las redes estáticas de comportamiento dinámico. Este hecho hace que algunos autores las incluyan dentro de las redes estáticas, aunque en esta memoria hemos preferido considerarlas como un tipo de redes diferente.

Como segunda clase de redes neuronales debemos nombrar las redes neuronales dinámicas. Estas a diferencia de las anteriores permiten establecer una relación entre salidas y entradas y/o salidas y entradas previas. Esto añade cierta memoria a estas redes. Matemáticamente, esta memoria se traduce en la aparición de ecuaciones diferenciales o ecuaciones en diferencia formando parte del modelo de las mismas. Como ejemplos de este tipo de redes encontramos la red de Hopfield, la red de retardos en el tiempo (*Time Delay Neural Network*), la red de tiempo discreto (*Time Discrete Neural Network*), etc. Las redes neuronales dinámicas se han revelado útiles en problemas de modelización de la dinámica directa e inversa de sistemas complejos, tales como robots, cohetes, naves espaciales, etc., así como en la modelización de circuitos secuenciales y en la conversión de texto a voz.

Por último podemos destacar las denominadas redes auto-organizativas, en las cuales los nodos vecinos dentro de la red neuronal compiten en actividad por medio de las interacciones mutuas laterales, y evolucionan adaptativamente hacia detectores específicos de las diferentes señales de entrada. El aprendizaje en este caso se denomina aprendizaje competitivo, no supervisado o autoorganizativo.

Estas se han aplicado con éxito en problemas de reconocimiento de patrones, control de procesos e incluso en el procesamiento de información semántica.

1.5. Aplicaciones de las redes neuronales.

Por último, se muestran algunas de las múltiples aplicaciones que pueden darse a las redes neuronales en este campo.

Visión artificial: Se emplean modelos de redes neuronales que son capaces de emular características del funcionamiento visual humano permitiendo, por ejemplo, el reconocimiento de imágenes texturizadas en color, el aprendizaje para determinar posiciones a partir de la información proveniente de dos cámaras, y representación de la visión binocular.

Reconocimiento y categorización de patrones: Estas redes emplean las arquitecturas de la Teoría de la Resonancia Adaptativa o ART (Widrow, 1960). Entre otras aplicaciones se encuentran el reconocimiento de caracteres manuscritos, autorización de descubiertos bancarios y clasificación de cromosomas.

Procesos químicos: Dos aplicaciones posibles son: el control de la temperatura en un reactor químico y el control de procesos químico-orgánicos no lineales.

Control de motor: Permiten resolver el problema de cinemática inversa en manipuladores y robótica móvil, consistente en determinar la secuencia de movimientos que deben realizar las distintas partes del robot para alcanzar una posición deseada. También permiten el aprendizaje de la dinámica del manipulador, es decir, de la generación de las fuerzas y pares que hay que aplicar para producir un movimiento determinado.

Otros campos, como la predicción económica y problemas de gestión, aprendizaje preventivo, etcétera.

1.6. El futuro de las redes neuronales

Las redes neuronales alcanzan cada vez mayor auge, teniendo multitud de aplicaciones en campos diversos y dando soluciones sencillas a problemas cuya resolución resulta complicada cuando se emplean máquinas algorítmicas. Aún así, el futuro de las redes neuronales no está todavía claro y será en los próximos años cuando se determine su evolución.

CAPÍTULO II

APRENDIZAJE DE LAS REDES NEURONALES ARTIFICIALES.

2.1 Introducción.

Las redes neuronales artificiales han sido estudiadas desde hace cuatro décadas cuando Rosenbalt aplicó por primera vez un perceptrón de una sola capa a un problema de aprendizaje de clasificación de patrones. El interés de las Redes Neuronales (*RNA*) disminuyo en los años 70, pero volvió a resurgir con la aparición de nuevos algoritmos de aprendizaje, circuitos *VLSI* y técnicas de procesamiento paralelo. Es este un campo de extenso desarrollo que no pretendemos abordad en su totalidad en este trabajo. Nos centraremos en la redes con capacidad de aprendizaje supervisado. En este sentido, nos parece conveniente utilizar el concepto de red Adaptativa introducido por J.-S.R. Jang and Sun, C.T. (Jang y Sun, 1995) como superconjunto de todos los tipos de redes neuronales con capacidad de aprendizaje supervisado que, cierta manera, unifica las redes neuronales y los sistemas desde un punto de vista estructural.

2.2 Redes Neuronales Artificiales.

Las redes neuronales artificiales son redes de elementos simples usualmente adaptativos y de sus organizaciones jerárquicas masivamente en paralelo, que se pretende que interactúen con los objetos del mundo real de la misma manera que lo hacen los sistemas nerviosos biológicos. Esta definición de Teuvo Kohonen (Kohonen, 1988) da una idea de la estructura y pretensiones que caracterizan a las Redes Neuronales Artificiales (*RNA*). Estas redes pueden ser vistas como

algoritmos de computación de la información, capaces de realizar tareas de clasificación de la información, memoria asociativa y comprensión de la información. La programación de estos algoritmos se realiza mediante el aprendizaje, es decir, a través de la enseñanza con un conjunto de ejemplos bien definidos, en base a los cuales el sistema es capaz de generalizar el conocimiento adquirido.

2.3 Arquitectura de las Redes Neuronales Artificiales Adaptativas.

Una red Adaptativa es una estructura de red cuyo comportamiento global de entrada/salida se determina por los valores de una colección de parámetros modificables. Más específicamente, la configuración de estas redes está compuesta de un conjunto de nodos interconectados por conexiones directas, donde cada nodo es una unidad de proceso que produce una función de nodo de sus señales de entrada para generar una única señal de salida, donde cada conexión especifica la dirección del flujo de señales. Usualmente, una función nodal es una función parametrizada de parámetros modificables; cambiando estos parámetros cambiamos la función nodal y el comportamiento general de la red.

2.4 El Perceptron Simple.

En 1957, Frank Rosenblatt presentó el **Perceptron**, una red neuronal con aprendizaje supervisado cuya regla de aprendizaje era una modificación de la propuesta por Hebb. El Perceptron trabaja con patrones de entrada binarios, y su funcionamiento, por tratarse de una red supervisada, se realiza en dos fases: una primera en la que se presentan las entradas y la salidas deseadas; en esta fase la red aprende la salida que debe dar para cada entrada.

La principal aportación del Perceptron es que la adaptación de los pesos se realiza teniendo en cuenta el error entre la salida que da la red y la salida que se desea, como se muestra en la figura 2.1. En la fase siguiente, de operación, la red ***es capaz*** de responder adecuadamente cuando se le vuelven a presentar los datos de entrada. Se crearon grandes expectativas sobre sus aplicaciones, que posteriormente se tornaron en gran decepción cuando en 1969 Minsky y Papert demostraron las grandes limitaciones de esta red.

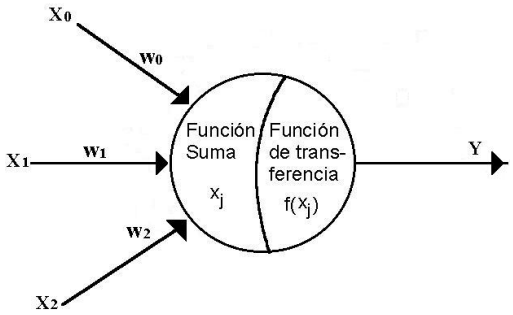


Figura 2.1- El perceptron simple

2.5. Estructura

Un Perceptron consta de dos niveles o capas (figura 2.1). El primer nivel está formado por un número de X_i entradas, denominadas unidades sensoriales, en la figura 2 son dos entradas (X_i) y el umbral (X_0). El segundo nivel está compuesto por un número de salidas, denominadas unidades de asociación, cuyas entradas son las salidas de las unidades de entrada de la adaptación de los pesos (w_i). En la figura 4 sólo se empleará una neurona en la capa de salida. Las unidades de entrada tienen una sola entrada correspondiente a una de las entradas a la red, y una sola salida. Estas unidades transmiten la señal que aparece en su entrada.

2.6. Aprendizaje del Perceptron Simple.

A continuación se muestran los pasos para el algoritmo general para el aprendizaje del Perceptron simple.

1.- Asignación de valores aleatorios de los pesos w_i y el umbral ($-w_0 = \theta$).

2.- Presentación de un nuevo par de entrada-salida deseada.

3.- Cálculo de la salida actual mediante la ecuación siguiente:

$$y(t) = f\left[\sum_i w_i(t)x_i(t) - \Theta\right] \tag{2.1}$$

donde :

f(x): Es la señal de activación,

w_i: Son los pesos, y

X_i: Son las entradas.

Θ: Umbral

4.- Adaptación de los pesos esta dada por la siguiente ecuación.

$$w_i(t+1) = w_i(t) + \alpha\left[Y_d(t) - Y(t)\right]x_i(t) \tag{2:2}$$

Donde:

X_i: Son las entradas,

Y_d: Salida deseada, y

α : Es un factor de ganancia (aprendizaje) con un rango de valores de $0 \leq \alpha \leq 1$

Repetir la adaptación de los pesos hasta que el error sea igual a cero.

5.- Iniciar en el paso 2.

La demostración del algoritmo de aprendizaje para el perceptron simple de las compuertas lógicas básicas las compuertas lógicas AND y OR, observaron que los resultados obtenidos, confirmaron el aprendizaje de ésta red neuronal con éstas compuertas.

En los años 60 se propusieron otros dos modelos, también supervisados, basados en el Perceptron de Rosenblatt denominados **Adaline** y **Madaline**. En estos, la adaptación de los pesos se realiza teniendo en cuenta el error, calculado como la diferencia entre la salida deseada y la dada por la red, al igual que en el Perceptron. Sin embargo, la regla de aprendizaje empleada es distinta. Se define una función error para cada neurona que da cuenta del error cometido para cada valor posible de los pesos cuando se presenta una entrada a la neurona. Así, la regla de aprendizaje hace que la variación de los pesos se produzca en la dirección y sentido contrario del vector gradiente del error. A esta regla de aprendizaje se la denomina Delta.

La era moderna de las redes neuronales artificiales surge con la técnica de aprendizaje de propagación hacia atrás o ***Back Propagation*** (Cui y Shin, 1992). La estructura de las redes citadas anteriormente (Perceptron Simple, Adaline y Madaline) consta de dos capas: una capa primera formada por unidades que dejan pasar la entrada y que no tienen aprendizaje, y una segunda capa formada por una o varias neuronas en el caso del Madaline. La contribución de Minsky y Papert fue la de demostrar que una red del tipo Perceptron simple no es capaz de aprender todas las posibles combinaciones entre entradas y salidas. La solución del problema consiste en añadir capas intermedias de neuronas, como se muestra en la figura 2.2, introduciendo de esta forma el problema de cómo enseñar a estas capas intermedias.

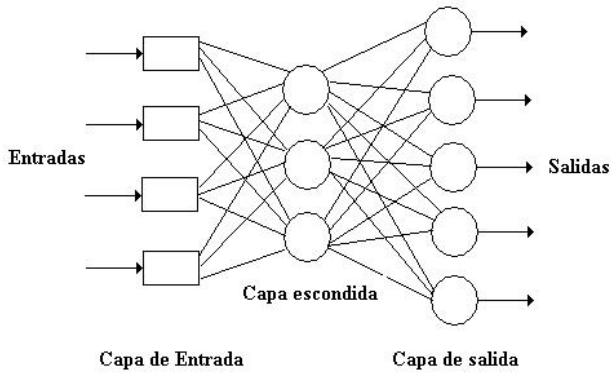


Figura 2.2- Modelo de un Perceptron con tres capas de neuronas.

En la figura 2.2 podemos observar que la red neuronal contiene tres capas de neuronas que son: capa de entrada, capa escondida y capa de salida. En algunos textos la capa de entrada no se cuenta, pero del punto de vista de la red, nosotros la tomaremos como una más.

2.7 Algoritmo de Retropropagación

El algoritmo de retropropagación, junto con sus variantes, es el método de aprendizaje mas utilizado en el entrenamiento de las redes adaptativas. La retropropagación se fundamenta en el conocido método del gradiente para la minimización de funciones en espacios multidimensionales. La parte central de una regla de aprendizaje para una red adaptativa consiste en cómo obtener recursivamente el vector gradiente en el que cada elemento se define como la derivada de una medida de error con respecto a un parámetro. Esto se hace por medio de la regla de la cadena, y el método se denomina regla de aprendizaje de retropropagación porque el vector gradiente se calcula en la dirección opuesta al flujo de la salida en cada nodo. Una vez calculado el gradiente, existen diversas técnicas de optimización y regresión para actualizar los parámetros de la red. Puede encontrarse una detallada y muy clara exposición de los métodos de optimización y búsqueda de mínimos de funciones y de su aplicación en el entrenamiento de una red neuronal artificial. Siendo el algoritmo de retropropagación y sus variantes los más utilizados en el entrenamiento de las redes adaptativas.

2.8. Redes Artificiales de Retropropagación.

Las redes de retropropagación son aquellas en las que el ajuste de los pesos durante la fase de entrenamiento se realiza invirtiendo la dirección del flujo de la activación en la red mediante una regla llamada "de retropropagación", que ajusta cada peso para hacer mínimo el error de las unidades de salida.

Las redes en las que se puede aplicar esta regla se caracterizan por lo siguiente:

a. Tienen dos o más capas
b. Son de "flujo directo", es decir, el patrón de conectividad de las unidades no presenta recurrencia o retroalimentación.
c. Utilizan una función de activación continúa y creciente. La más habitual es la función sigmoide computada a partir de la habitual función aditiva ponderada.

De este modo, lo que realiza una red de retropropagación es siempre una proyección (*"mapping"*) del vector de entrada en el de salida. Las redes con estas características han mostrado ser capaces de realizar muy diversos tipos de proyecciones (es decir, funciones entrada-salida) con gran fiabilidad. La regla de retropropagación es la principal regla de aprendizaje instruido en las redes neuronales actuales. En líneas generales, permite que en cada ejemplo la red compare su salida con la salida "correcta", y reajuste los pesos de sus conexiones en función del resultado de dicha comparación. La red neuronal artificial que ha sido usada para la identificación de procesos dinámicos y su control, es la red neuronal conocida como ***red de propagación hacia a atrás o Retropropagación.*** (Widrow, y McClelland, 1986) La red de la figura 5 es un ejemplo típico de una red de propagación hacia atrás, es aquí donde este método tiene una gran importancia. En éste comparamos la salida real con la salida deseada. La diferencia entre ambas constituye un error que se propaga hacia atrás desde la capa de salida hasta la de entrada permitiendo así la adaptación de los pesos de las neuronas intermedias mediante un método de aprendizaje.

Este método no el único existente, pero puede afirmarse que este método y sus modificaciones, ha sido de lo más exitoso para la solución una gran diversidad de aplicaciones prácticas. El método propuesto por Widrow y Hoff (Widrow y Hoff, 1960) fue generalizado para el caso de redes neuronales multicapas llamado ***algoritmo de propagación hacia atrás*** (Werbos, 1974, Rumelhart y McClelland,

1986). Una característica muy importante de las redes multicapas, es el número con las que cuenta. Las redes multicapas pueden obtener resultados que son imposibles para redes de dos capas. Como ha sido discutido por Rumelhart y McClelland en 1986, la adición de capas escondidas al algoritmo de propagación hacia atrás permite desarrollar una representación interna del problema que puede ser vital para la solución de éste. La red de propagación hacia atrás es una red neuronal que es capaz de obtener una aproximación muy fina para cualquier función no lineal y = f(x), a partir de cualquier conjunto de pares de datos x, y. Posteriormente se han desarrollado otros modelos que permiten un aprendizaje no supervisado como el mapa auto-organizativo de Kohonen, los basados en la Teoría de Resonancia Adaptativa (*ART*) de Grossberg y Carpenter, o los modelos de control motor de Bullock, Gaudiano y Grossberg, entre otros.

2.9. Aprendizaje de retropropagación del Perceptron con tres capas de neuronas.

A continuación se muestran los pasos para el algoritmo general simplificado para el aprendizaje de retropropagación del Perceptron con tres capas de neuronas, considerando el modelo de la red neuronal de la figura 2.3 y supongamos que tenemos un conjunto de pares de datos x^p, y^p, p = 1,2,3,...,N.

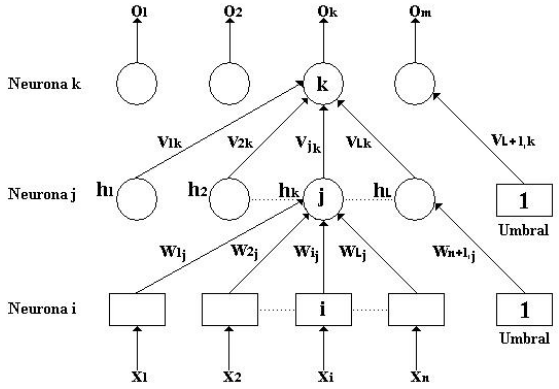


Figura 2.3.-Modelo de una red neuronal artificial de 3 capas.

Para capa oculta de la neurona j, definimos la función de excitación, para una muestra de datos P, con la siguiente ecuación:

$$S_j = \sum_{i=1}^{n} w_{ij}{}^{P} x_j + w^{P}{}_{(n+1)} j \tag{2.3}$$

Para tomar el rango de 0 a 1 o de 1 a –1 , necesitamos conocer como se comportan los datos de entrada y salida, x_i, y_i, al vez estos datos deben de estar normalizados en los rangos anteriores. De la figura 6 los coeficientes w_{ij}, v_{jk}, son conocidos como los ***pesos*** de la red neuronal, a la entrada n + 1, j, observamos que el valor es uno, es conocida como ***bias o umbral***, por lo que esta entrada permite que la red pueda modelar sistemas, en donde las salidas pueden tener valores distintos de cero, cuando las entradas son cero. La salida de la capa oculta de la neurona j es calculada generalmente por una función no-lineal de la excitación, en algunos casos también es obtenida con una función lineal. Ahora definiremos la función no-lineal de salida o función de activación, como la función sigmoide como se muestra en la figura 2.4, para la salida de la neurona j y la neurona k.

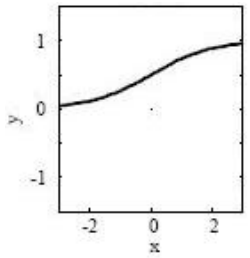


Figura 2.4.- Función de activación (Sigmoide).

La función sigmoide es la más utilizada en las redes neuronales, debido a sus grandes ventajas dada su simplicidad, pero pueden utilizarse otras funciones de activación como: la tangente hiperbólica, la campana de Gauss, saturación, etc. A continuación en la figura 2.5 se muestran estas señales de activación.

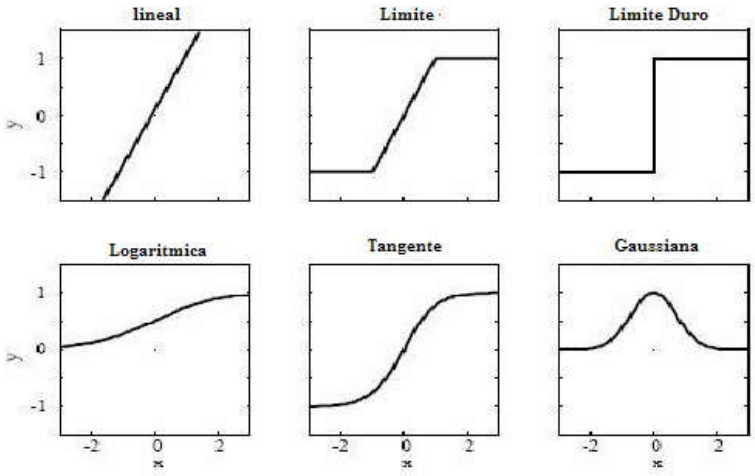


Figura 2.5.- Señales de activación para las redes neuronales artificiales.

La salida de la capa oculta de la neurona j esta dada por la siguiente expresión:

$$h_j^P = f\left(S_j^P\right) = \frac{1}{1+e^{-S_j^P}} \tag{2.4}$$

La excitación de la capa de salida de la neurona k, la obtenemos de forma análoga, por la siguiente expresión.

$$r_k^P = \sum_{j=1}^{L} v_{jk}{}^P h_j^P + v_{(L+1)k}^P \tag{2.5}$$

Para la salida de la capa de salida de la neurona k, queda expresada por la siguiente ecuación:

$$O_k^P = f(r_k^P) = \frac{1}{1+e^{-r_k^P}} \tag{2.6}$$

El error en la salida de la neurona k, es definido para una muestra de datos P como sigue:

$$e_k^P = y_k^P - O_k^P \tag{2.7}$$

Donde:

y_k^P : salida deseada, y

O_k^P : salida de la red.

El criterio de minimización para una muestra de datos P queda es expresada por:

$$Ep = \frac{1}{2}\sum_{k=1}^{m}\left(e_k^P\right)^2 = \frac{1}{2}\sum_{k=1}^{m}\left(y_k^P - O_k^P\right)^2 = \frac{1}{2}\sum_{k=1}^{m}\left(y_k^P - f(r_k^P)\right)^2 \tag{2.8}$$

Cuando la expresión anterior es mínima, y_k es igual a la salida O_k, la expresión quedaría de la siguiente forma:

$$\nabla Ep = \begin{bmatrix} \dfrac{\partial Ep}{\partial v_{jk}^P} \\ \dfrac{\partial Ep}{\partial w_{ij}^P} \end{bmatrix} \tag{2.9}$$

Resolviendo el gradiente parte por parte, primero la derivada parcial y aplicando la regla de cadena de la derivada obtenemos la expresión:

$$\frac{\partial Ep}{\partial v_{jk}} = \frac{\partial Ep}{\partial O_k^P}\frac{\partial O_k^P}{\partial r_k^P}\frac{\partial r_k^P}{\partial v_{jk}} = -(y_k^P - O_k^P)O_k^P(1-O_k^P)h_j^P \tag{2.10}$$

Si hacemos:

$$\delta_k^P = (y_k^P - O_k^P)O_k^P(1-O_k^P)h_j^P$$

La ecuación (2.10) se simplifica como:

$$\frac{\partial Ep}{\partial v_{jk}} = \delta_k^P h_j^P \tag{2.11}$$

Empleamos el mismo método para la segunda parcial obtenemos:

$$\frac{\partial Ep}{\partial w_{ij}} = \left[\sum_{k=1}^{m}\left(\frac{\partial Ep}{\partial O_k^P}\frac{\partial O_k^P}{\partial r_k^P}\frac{\partial r_k^P}{\partial h_j}\right)\right]\frac{\partial h_i^P}{\partial S_j^P}\frac{\partial S_j^P}{\partial w_{ij}} = \sum_{k=1}^{m}\left[-(y_k^P - O_k^P)O_k^P(1-O_k^P)v_{jk}\right]h_j^P(1-h_j^P)x_j$$

Si hacemos:

$$\Delta_j^P = \left(\sum_{k=1}^{m} \delta_k^P v_{jk}^P \right) h_j^P (1 - h_j^P)$$

La ecuación (2.11) se simplifica como:

$$\frac{\partial Ep}{\partial w_{ij}} = -\Delta_j^P x_j \tag{2.12}$$

Por lo tanto el gradiente quedaría determinado por:

$$\nabla Ep = -\begin{bmatrix} \delta_k^P h_k^P \\ \Delta_j^P x_j^P \end{bmatrix} \tag{2.13}$$

El entrenamiento de las neuronas de una red de tres capas por el algoritmo de propagación hacia atrás, consiste en aplicar el método del descenso más rápido, es decir moverse en la dirección negativa del gradiente, para ir actualizando los coeficientes de los valores de los pesos w_{ij}, v_{jk}, con cada par de vectores de entrada y salida x_p, y_p, es decir, el gradiente en su forma recursiva quedaría expresada con la siguiente expresión:

$$\begin{bmatrix} v_{jk}^P \\ w_{ij}^P \end{bmatrix} = \begin{bmatrix} v_{jk}^{P-1} \\ w_{ij}^{P-1} \end{bmatrix} - \eta \Delta Ep \tag{2.14}$$

De la ecuación (2.14) la constante eta (η), es denominada coeficiente de aprendizaje, es el tamaño con el que nos movemos en el gradiente, por lo que su rango esta entre 0 y 1, para que así, podamos obtener la convergencia, si es mayor a uno diverge.

Sustituyendo la ecuación (2.14) en la ecuación (2.13) obtenemos la siguiente expresión:

$$\begin{bmatrix} v_{jk}^{P} \\ w_{ij}^{P} \end{bmatrix} = \begin{bmatrix} v_{jk}^{P-1} \\ w_{ij}^{P-1} \end{bmatrix} - \eta \begin{bmatrix} \delta_{k}^{P} h_{k}^{P} \\ \Delta_{j}^{P} x_{j}^{P} \end{bmatrix} \tag{2.15}$$

El método de aprendizaje de la red neuronal de tres capas consiste en presentar secuencialmente las entradas x_i^P para i = 1,2,3.....,n y P = 1,2,3,....,N, después calcular las salidas de la red O_k^P para k = 1,2,3.....,m y P = 1,2,3,....,M, los errores e_k^P, el criterio de minimización Ep, aplicando algún procedimiento de minimización de la función Ep con respecto a los coeficientes de los pesos w_{ij}, y v_{jk}, de manera que estos se vayan aproximando a unos valores que garanticen un error muy pequeño entre la salida de la red O_k^P y los datos de la salida y_k.

El método de aprendizaje anterior se repite, hasta ciclarse, y deja de ciclarse cuando el error en cada salida y el criterio Ep, estén por debajo del límite prefijado. Logrando esto, decimos que la red esta entrenada, es decir, la red es capaz de reproducir la función y = f(x) con gran exactitud de todos los datos que han sido bien seleccionados y suficientes. A cada ciclo realizado por el método de repropagación hacia atrás se le denomina **época** en el argot de las redes neuronales.

2.10. Red Neuronal Artificial Autoajustable

La idea general de esta red neuronal artificial autoajustable directa es entrenar esta red con el algoritmo de aprendizaje con la propagación hacia atrás, el error de regulación en lugar del error de salida de la red. Con las ideas de Cui y Shin proponemos un controlador neuronal, con la característica principal de que el entrenamiento previo de la red es sustituido por el ajuste permanente de los coeficientes de peso basado en el error de regulación.

La red neuronal artificial autoajustable (*RNAA*) propuesta para el control de una planta dinámica se muestra en el diagrama de control de lazo cerrado de la figura 2.6.

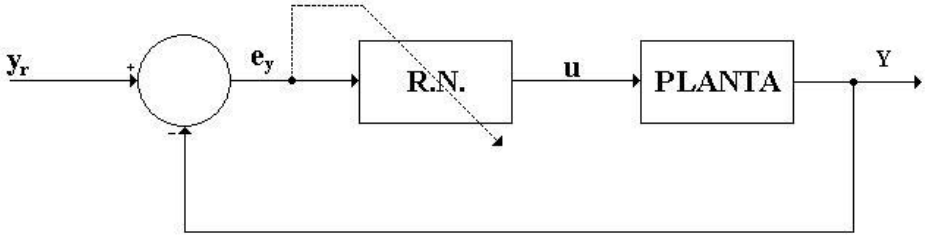


Figura 2.6.- Esquema del control de una planta con una red neuronal Artificial Autoajustable.

La red de neuronal artificial autoajustable (*RNAA*) del esquema del control de la figura 2.6, consiste de un Perceptron multicapa con tres capas de neuronas, una de entrada, una salida y una escondida. La estructura interna de la red neuronal artificial autoajustable se muestra en la figura 2.7.

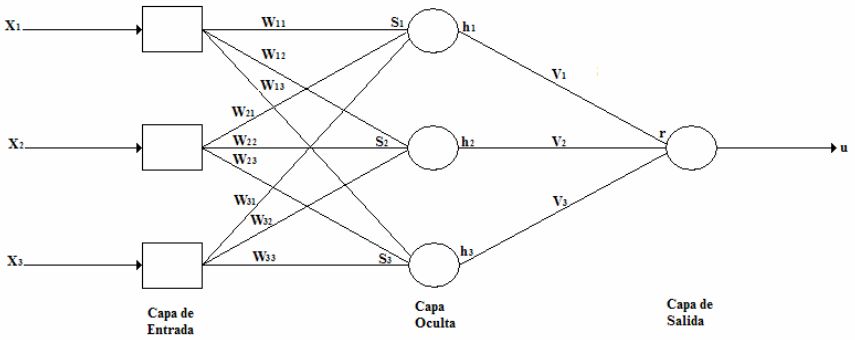


Figura 2.7.- Estructura interna de la Red Neuronal Artificial Autoajustable.

Se observa de la figura 2.7 que la capa de salida consta de una sola neurona, porque estamos limitando el análisis para el caso de procesos de una entrada y una salida. En donde los coeficientes son ajustados utilizando el algoritmo de aprendizaje de retropropagación o propagación hacia atrás del error.

Para este caso, en lugar del error de salida de la red:

$$e_u = u_d - u \tag{2.16}$$

Utilizaremos el error de salida del proceso:

$$e_y(t) = y_r(t) - y(t) \tag{2.17}$$

2.11 Algoritmo de aprendizaje de la red neuronal artificial autoajustable.

Los pasos del algoritmo de aprendizaje de retropropagación o propagación hacia atrás de la red neuronal autoajustable de la figura 2.7, se enlistan a continuación:

1) Inicializar los pesos de la red neuronal autoajustable *Vj* y *Wij* con valores pequeños.
2) Presentar un patrón de datos de entrada X, especificando el error de la red neuronal autoajustable y del proceso como sigue:

$$x(t) = \left[x(t), x\,(t-1), x(t-2)\,\right] \tag{2.18}$$

3) Realizar el calculo de la entrada de la neurona de la capa escondida, la cual esta determinada por la siguiente ecuación:

$$S_j = \sum_{i=1}^{3} w_{ji}\; x_i \;, i = 1,2,3. \tag{2.19}$$

4) Se utiliza la función de activación, para calcular la salida de la neurona de la capa escondida, que esta determinada por la siguiente ecuación:

$$h_j = \frac{1}{1 + e^{-S_j}} \tag{2.20}$$

5) Se calcula r que es la multiplicación de la salida de la neurona de la capa escondida por los peso *Vj* y esta determinada por la siguiente ecuación:

$$r = \sum_{j=1}^{3} v_j \; h_j \tag{2.21}$$

6) El calculo de la salida de la neurona de la capa de salida se determina con la siguiente ecuación:

$$u(t) = \frac{1}{1+e^{-r}} \tag{2.22}$$

7) La actualización de los pesos *Wji* y *Vj* se determina por las siguientes ecuaciones:

$$v_j(t+1) = v_j(t) + \eta\delta_1 h_j \tag{2.23}$$

$$w_{ji}(t+1) = w_{ji}(t) + \eta\delta_2 x_i \tag{2.24}$$

Donde: $\delta_1 = (e_y)\left[u(1-u)\right]$

$$\delta_2 = \delta_1 v_j h_j (1-h_j)$$

2.12.- Ejemplo usando el algoritmo de aprendizaje de Retropropagación

Sea la red de la figura 2.8. Calcular la actualización de los pesos *Vi* y *Wji*

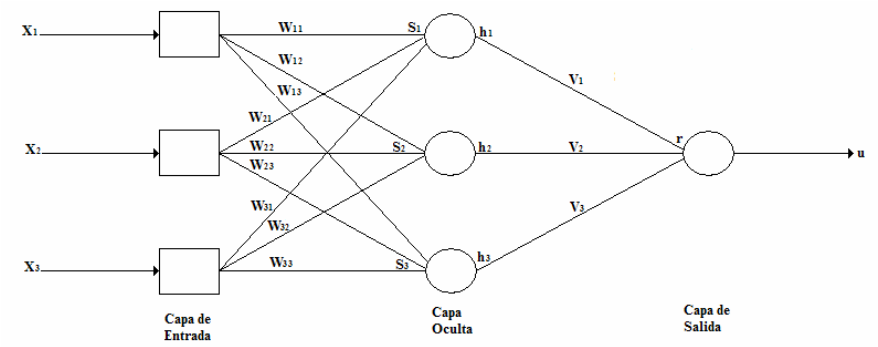


Figura 2.8.- RNAA

Inicializamos todos los pesos Vj y Wji con un valor pequeño de 0.1
Para un patrón de entradas de X1 = 0.5, X2 = 0.7 y X3 = 1.

De acuerdo a la figura 2.8, se calcula la entrada a la neurona de la capa escondida que es parámetro *S*.

Utilizando la ecuación (2.19), se realiza el cálculo de los parámetros *S*

$$S_1 = 0.5(0.1) + 0.7(0.1) + 1(0.1) = 0.22$$
$$S_2 = 0.5(0.1) + 0.7(0.1) + 1(0.1) = 0.22$$
$$S_3 = 0.5(0.1) + 0.7(0.1) + 1(0.1) = 0.22$$

Ahora se realiza el cálculo de los parámetros *hj* que es la salida de la capa escondida utilizando la función de activación, descrita en la ecuación (2.20)

$$h_1 = \frac{1}{1+e^{-0.22}} = \frac{1}{1+e^{-0.22}} = \frac{1}{1+0.8025} = \frac{1}{1.8025} = 0.5547$$
$$h_2 = \frac{1}{1+e^{-0.22}} = \frac{1}{1+e^{-0.22}} = \frac{1}{1+0.8025} = \frac{1}{1.8025} = 0.5547$$
$$h_3 = \frac{1}{1+e^{-0.22}} = \frac{1}{1+e^{-0.22}} = \frac{1}{1+0.8025} = \frac{1}{1.8025} = 0.5547$$

El siguiente paso es calcular la entrada de la neurona de salida de la capa de salida, es decir, el parámetro *r*. Esto se realiza con la ecuación (2.21) descrita anteriormente.

$$r = (0.1)(0.5547) + (0.1)(0.5547) + (0.1)(0.5547) = 0.16641$$

Para calcular la salida de la neurona de la capa de salida, se utiliza nuevamente la función de activación descrita en la ecuación (2.22)

$$u(t) = \frac{1}{1+e^{-0.16641}} = \frac{1}{1+0.8466} = \frac{1}{1.8466} = 0.5415$$

Ahora se calculan los pesos Vj con la siguiente ecuación:

$$v_j(t+1) = v_j(t) + \eta\delta_1 h_j$$

Donde:

$$\delta_2 = \delta_1 v_j h_j (1 - h_j)$$

Procedemos a calcular δ_1 para un error de 0.1.

$$\delta_1 = (0.1)\left[0.5415(1-0.5415)\right] = 0.02482$$

Se realiza el cálculo de los pesos Vj.

$$v_1 = 0.1 + (1)(0.02482)(0.5547) = 0.11377$$
$$v_1 = 0.1 + (1)(0.02482)(0.5547) = 0.11377$$
$$v_1 = 0.1 + (1)(0.02482)(0.5547) = 0.11377$$

Ahora se calculan los pesos Wji con la siguiente ecuación:

$$w_{ji}(t+1) = w_{ji}(t) + \eta\delta_2 x_i$$

Donde:

$$\delta_2 = \delta_1 v_j h_j (1 - h_j)$$

Procedemos a calcular δ_2 para un error de 0.1.

$$\delta_2 = (0.02482)(0.1)\left[0.5547(1-0.5547)\right] = 0.000613$$

Se realiza el cálculo de los pesos Wji

$$w_{11} = 0.1 + (1)(0.000613)(0.5) = 0.1003065$$
$$w_{12} = 0.1 + (1)(0.000613)(0.7) = 0.1004291$$
$$w_{13} = 0.1 + (1)(0.000613)(1) = 0.100613$$
$$w_{21} = 0.1 + (1)(0.000613)(0.5) = 0.1003065$$
$$w_{22} = 0.1 + (1)(0.000613)(0.7) = 0.1004291$$
$$w_{23} = 0.1 + (1)(0.000613)(1) = 0.100613$$
$$w_{31} = 0.1 + (1)(0.000613)(0.5) = 0.1003065$$
$$w_{32} = 0.1 + (1)(0.000613)(0.7) = 0.1004291$$
$$w_{33} = 0.1 + (1)(0.000613)(1) = 0.100613$$

.

Ahora con los nuevos pesos Wji y Vj se repiten todos los cálculos anteriores.

CAPÍTULO III.
DESCRIPCIÓN DEL ALGORITMO DE APRENDIZAJE EN VHDL.

3.1. Introducción.

Este capítulo presenta la descripción del algoritmo de aprendizaje de la red neuronal autoajustable artificial autoajustable en el software de diseño de circuitos programables VHDL. Se presenta primero la descripción del problema y posteriormente la propuesta de solución que se tiene hasta el momento por lo tanto este capítulo queda organizado de la siguiente manera:

3.2 Descripción del problema
3.3 Propuesta de solución
3.4 Solución aplicando el software de diseño de circuitos programables VHDL
3.4.1 Metodología
3.5 Descripción del algoritmo de aprendizaje de la red neuronal artificial autoajustable en VHDL
3.6 Conclusiones

3.2. Descripción del problema.

La red neuronal artificial autoajustable (*RNAA*) propuesta para implementar el modelado del algoritmo de aprendizaje de Retropropagación en un FPGA se muestra en la figura 3.1.

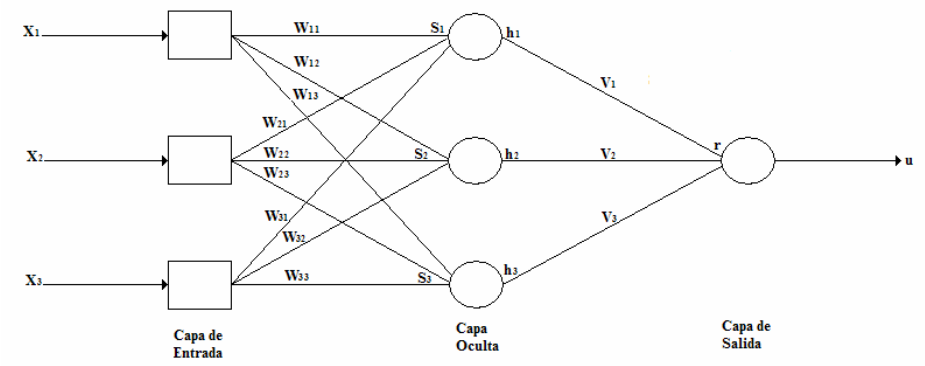


Figura 3.1.- red neuronal autoajustable.

A continuación se describen las ecuaciones digitalmente en forma general del algoritmo de propagación hacia atrás o retropropagación con su bloque lógicos:

1) Inicializar los pesos de la red neuronal autoajustable *Vj* y *Wij* con valores pequeños.
2) Presentar un patrón de datos de entrada X, especificando el error de la red neuronal autoajustable y del proceso como sigue:

$$x(t) = \left[x(t), x\ (t-1), x(t-2)\ \right] \tag{3.1}$$

3) Realizar el calculo de la entrada de la neurona de la capa escondida, la cual esta determinada por la siguiente ecuación:

$$S_j = \sum_{i=1}^{3} w_{ji}\ x_i\ , i = 1,2,3. \tag{3.2}$$

4) Se utiliza la función de activación, para calcular la salida de la neurona de la capa escondida, que esta determinada por la siguiente ecuación:

$$h_j = \frac{1}{1+e^{-S_j}} \tag{3.3}$$

5) Se calcula r que es la multiplicación de la salida de la neurona de la capa escondida por los peso *Vj* y esta determinada por la siguiente ecuación:

$$r = \sum_{j=1}^{3} v_j \; h_j \tag{3.4}$$

6) El calculo de la salida de la neurona de la capa de salida se determina con la siguiente ecuación:

$$u(t) = \frac{1}{1+e^{-r}} \tag{3.5}$$

7) La actualización de los pesos *Wji* y *Vj* se determina por las siguientes ecuaciones:

$$v_j(t+1) = v_j(t) + \eta\delta_1 h_j \tag{3.6}$$

$$w_{ji}(t+1) = w_{ji}(t) + \eta\delta_2 x_i \tag{3.7}$$

Donde:

$$\delta_1 = (e_y)\left[u(1-u)\right]$$

$$\delta_2 = \delta_1 v_j h_j (1-h_j)$$

El problema consiste en como representar los pasos del algoritmo de aprendizaje de retropropagación de la red neuronal artificial autoajustable, se propone representar cada paso de este algoritmo en bloques lógicos, para obtener un mejor desarrollo en la descripción de cada uno de los pasos en la descripción de estas ecuaciones en VHDL.

3.3 Propuesta de Solución.

Un primer enfoque para la solución de este problema consistió en representar los pasos del algoritmo de aprendizaje de retropropagación en bloques lógicos mediante el siguiente método:

1) Representación en bloques lógicos de los pasos del algoritmo de aprendizaje de retropropagación
2) Representación interna de cada bloque lógico, en donde su estructura interna de cada bloque corresponderá a la representación de las ecuaciones en forma digital.
3) Realizar la unión de todos los bloque lógicos del algoritmo de aprendizaje de retropropagación en uno solo.
4) Realizar la simulación de todo el modelado del algoritmo de aprendizaje de retropropagación de la red neuronal artificial autoajustable en VHDL.
5) Realizar la síntesis del modelado del algoritmo de aprendizaje de retropropagación en el software de síntesis Xilinx ISE 8.2i.

3.4 Solución aplicando VHDL

3.4.1 Metodología.

La utilización de redes neuronales se ha difundido ampliamente en varias tareas y procesos de uso cotidiano tales como OCRs, reconocedores del habla, detectores de gases, etc. Aún más, versiones digitales de casi todas estas redes se pueden encontrar con facilidad en el mercado bajo la forma de dispositivos electrónicos basados en algún circuito integrado genérico o en ASICs especializados programados específicamente para cumplir con alguna de las tareas especificas. Por otro lado, el diseño de una red neuronal artificial que cumpla cierta tarea particular y su posterior síntesis en un circuito digital demanda bastante tiempo y esfuerzo.

La herramienta desarrollada propone una metodología de diseño por pasos que permite al diseñador de la red neuronal artificial concentrarse en los aspectos estructurales propios de la red como la cantidad de entradas para este caso están definidas, el número de neuronas ocultas también están definidas, los valores de los pesos sinápticos, el uso de bias, etc. A partir de estos parámetros mínimos el

sistema genera la arquitectura necesaria para el funcionamiento de la red neuronal artificial al tiempo que genera el programa de control de la misma.

La red neuronal artificial autoajustable (*RNAA*) propuesta para la implementación de esta en VHDL se muestra en la figura 3.1. Para tener un mejor resultado en la descripción de los pasos del algoritmo de aprendizaje de propagación hacia atrás o retropropagación digitalmente, las ecuaciones que conforman los pasos del algoritmo se representaran en forma de bloques lógicos y cada uno de estos con su estructura interna, en donde cada bloque representara la ecuación de cada paso del algoritmo.

3.5 Descripción del algoritmo aprendizaje de la red neuronal Artificial Autoajustable en VHDL

Para el esquema de la red neuronal autoajustable representado en al figura 3.1 la entrada de datos será muestreada con una frecuencia de muestreo de un milisegundo (ms), y su representación digital para obtener la frecuencia de muestreo se muestra en la figura 3.2, el cuál se denomina bloque lógico de frecuencia de muestreo.

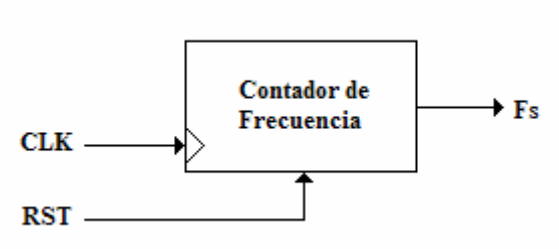


Figura 3.2.- Bloque lógico de frecuencia de muestreo

A continuación se describen las ecuaciones digitalmente en forma general del algoritmo de propagación hacia atrás o retropropagación con su bloque lógicos:

1) Inicializar los pesos de la red neuronal autoajustable *Vj* y *Wij* con valores pequeños. Los valores propuestos son de 0.1 para ambos pesos.

2) Presentar un patrón de datos de entrada X, a este bloque digital se designara Bloque Lógico de Entrada, donde la señal original tendrá un corrimiento de datos de dos tiempos anteriores, es decir, cada 1 ms, se tomara una muestra de un dato de la señal de referencia, para obtener el muestreo, se utiliza un contador con habitación para que vaya muestreando la señal de referencia que va estar en una memoria ROM, es decir guardada en una tabla exhaustiva de verdad, como se muestra en la figura 3.3.

$$x(t) = \left[x(t), x\,(t-1), x(t-2)\,\right] \tag{3. 1}$$

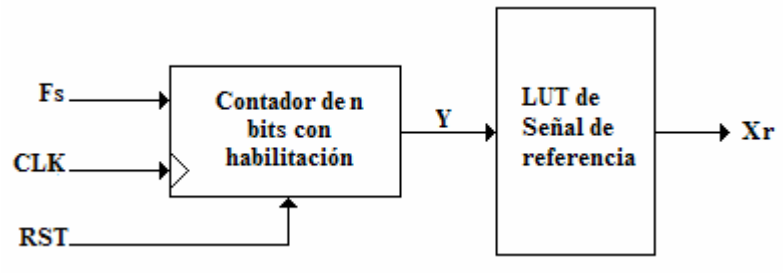


Figura 3.3.- Bloque Lógico de entrada (Señal de referencia)

Para la entrada de la red neuronal autoajustable en el algoritmo de aprendizaje la señal de referencia se desfasa dos tiempos anteriores para poder realizar la operación correspondiente como se ve en la ecuación (3.1), este bloque lógico se representa en forma digital en el bloque lógico de registro de corrimiento mostrado en la figura 3.4.

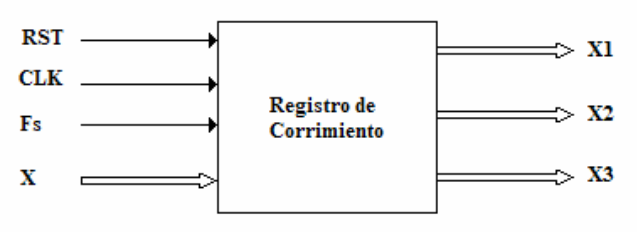


Figura 3.4.- Bloque Lógico de registro de corrimiento.

Para realizar este corrimiento en forma digital la estructura interna de este bloque, consta de tres registros para realizar la toma de muestra de datos como se muestra en la figura 3.5.

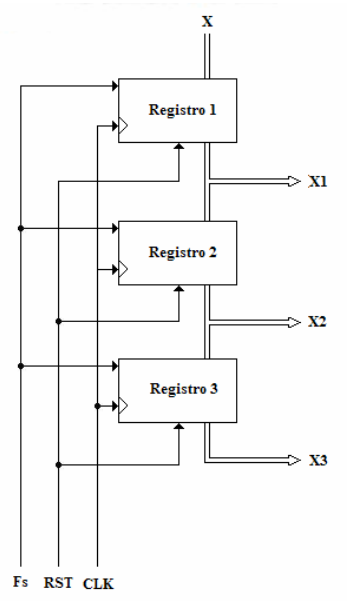


Figura 3.5.- Estructura interna del bloque lógico de corrimiento.

En este paso se procede a realizar el control para que la señal de referencia sea muestreada cada milisegundo, es decir, tomar un dato cada milisegundo. En la figura 3.6 se muestra el bloque de la máquina de control para la señal de referencia.

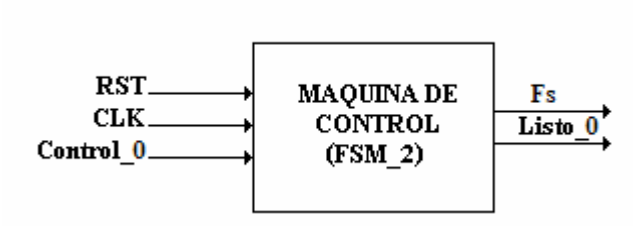


Figura 3.6.- Maquina de control de la señal de referencia.

En la figura 3.7.- Se muestra el grafo de la maquina de control, el cual determina las operaciones correspondientes.

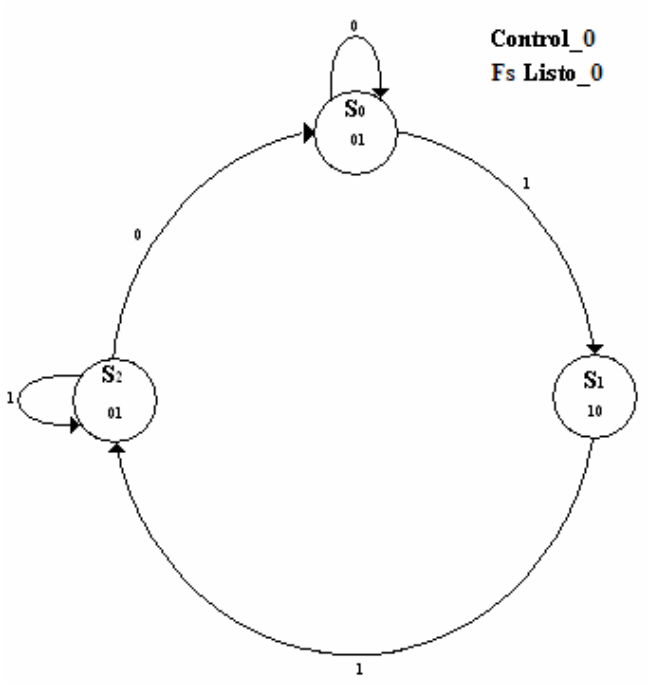


Figura 3.7.- Grafo de la maquina de control de la señal de referencia.

La máquina de control empieza en un estado S_0, donde la señal de *Control_0* y *Fs* se encuentran desactivadas, la señal de *Listo_0* se encuentra activada. Cuando la señal de *Control_0* se activa, la maquina de control pasa del estado S_0 al S_1, la señal de *Fs* se activa, se habilita el contador, el cual toma el primer dato de la señal de referencia, y la pasa al primer registro. La señal de *Listo_0* se desactiva. La señal de Control_0 se activa, la maquina de control pasa del estado

S_1 al S_2, la señal de *Fs* se desactiva, la señal de *Listo_0* se activa, regresando la maquina de control a su condición inicial, este proceso se repite esta que los dos siguientes registros tienen datos.

3) Realizar el calculo de la entrada de la neurona de la capa escondida, se denomina bloque lógico de la multiplicación de los parámetros de la entrada a la neurona de la capa escondida Sj.

$$S_j = \sum_{i=1}^{3} w_{ji}\ x_i\ , i = 1,2,3. \tag{3.2}$$

Donde j = 1, 2, 3:

Desarrollando la ecuación (3.2) obtenemos una serie de ecuaciones mostradas a continuación:

$$S_1 = X_1W_{11} + X_2W_{21} + X_3W_{31}$$
$$S_2 = X_1W_{12} + X_2W_{22} + X_3W_{32}$$
$$S_3 = X_1W_{13} + X_2W_{23} + X_3W_{33}$$

A continuación se muestra en la figura 3.8. El Bloque Lógico General correspondiente a las ecuaciones anteriores en forma digital.

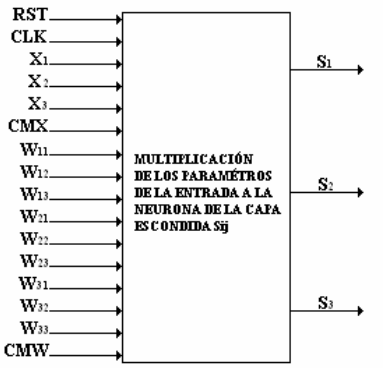


Figura 3.8.- Multiplicador acumulador de la entrada a la neurona de la capa escondida.

La estructura interna del bloque lógico de la multiplicación de los parámetros de la entrada a la neurona de la capa escondida es conocida como Multiplicador

Acumulador (*MAC*), la cual es la base de los sistemas de procesamiento digital de señales. Esta compuesta por dos multiplexores de entrada que son de 4 a 1 y de 12 a 1 de 18 bits cada entrada, los cuales van conectados al multiplicador acumulador donde se realiza la multiplicación de la entrada *X* y del peso *W*, la salida de este se conecta a un sumador el cual su salida va a un registro de acumulación y la salida de este es la otra entrada al sumador, la salida del sumador es de 38 bits al igual las entradas al sumador, en la salida del sumador se conecta a la vez con un registro de resultado en donde se muestra el resultado de la operación de la multiplicación de *S1*, así también encontramos otros dos registros conectados a la vez para los resultados de *S2* y *S3* estos son de 18 bits, como se muestra en la figura 3.9.

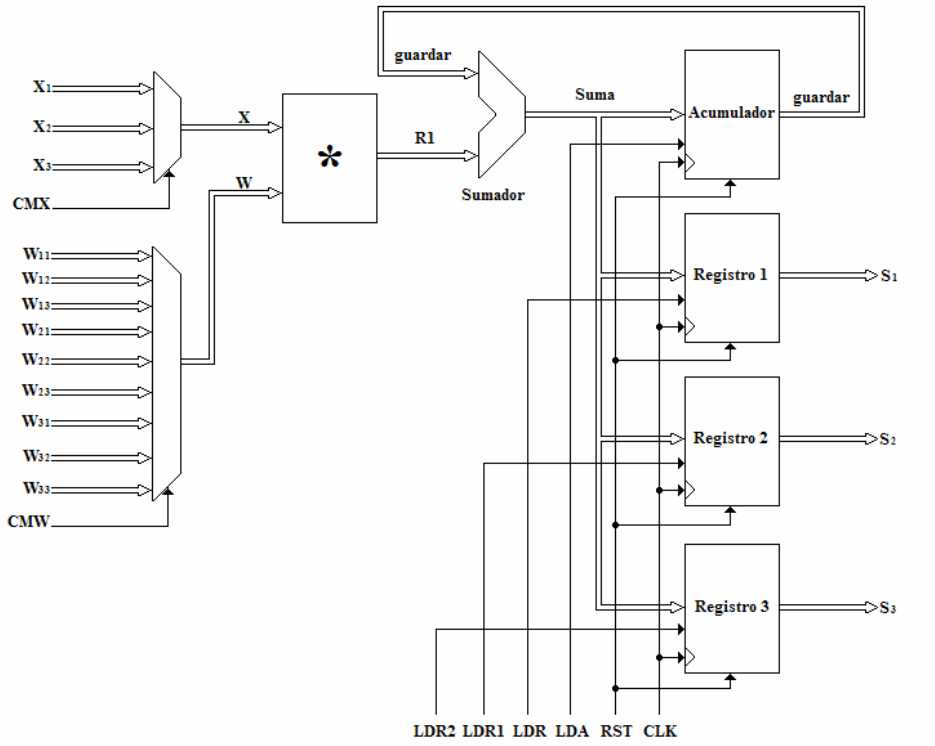


Figura 3.9.- Estructura interna del bloque lógico de la multiplicación de los parámetros de la entrada a la neurona de la capa escondida.

En la figura 3.10 se muestra el bloque del control de las operaciones que se realizan en el bloque lógico de la multiplicación de los parámetros de la entrada a

la neurona de la capa escondida utilizando la unidad MAC, así como su diagrama de Estados en la figura 3.11.

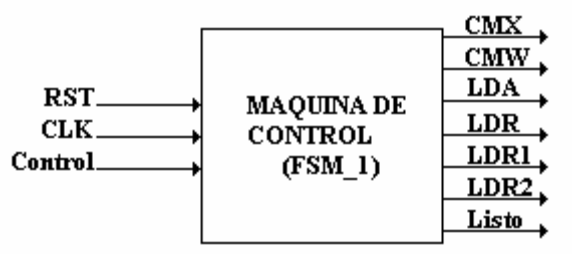


Figura 3.10.- Bloque general de la maquina de control de la MAC.

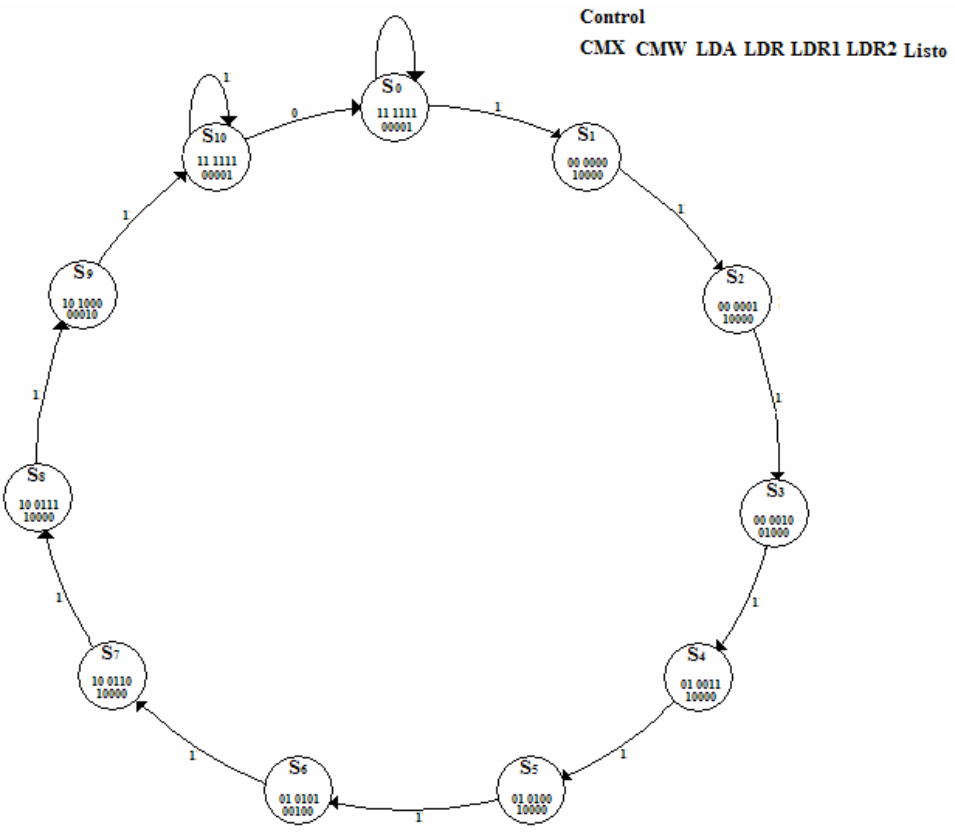


Figura 3.11.- Grafo de la maquina de control de la MAC

La maquina de estados del control que se observa en la figura 3.11, inicia en un estado S_0 donde espera la activación de la señal *Control* para iniciar la multiplicación, en este estado se borra el registro del acumulador y los registros

de resultados permanecen sin cambios y la señal *Listo* esta activa. Cuando la señal *Control* se activa el control pasa al estado S_1 el primer producto parcial se realiza; el registro del acumulador se encuentra cargando datos y los registros de resultados permanecen sin cambios, la señal *Listo* se encuentra desactivada. Cuando la señal de *Control* se activa otra vez la máquina de control pasa del estado S_1 pasa al estado S_2 el segundo producto parcial se efectúa; el registro del acumulador esta cargando datos y los registros de resultados permanecen sin cambios, la señal *Listo* se encuentra desactivada. Una vez más activada la señal de *Control*, la máquina de control pasa del estado S_2 al estado S_3, se realiza el tercer producto parcial, el registro del acumulador se borra y se presenta el resultado *S1* en el primer registro1, mientras los dos registros faltantes permanecen sin cambios, la señal *Listo* se encuentra desactivada. Esta operación de tres productos parciales y resultados se repite dos veces más para presentar los resultados *S2* y *S3*, es decir; que en el estado S_4 se realiza el primer producto parcial, el registro del acumulador se encuentra cargando datos, los registros de resultados permanecen sin cambios, la señal *Listo* se encuentra desactivada. En el estado S_5 se realiza el segundo producto parcial, el registro del acumulador se encuentra cargando datos, los registros de resultados permanecen sin cambios, la señal *Listo* se encuentra desactivada. En el estado S_6 se realiza el tercer producto parcial, el registro del acumulador se borra, el registro2 carga el resultado *S2*, los demás registros de resultados permanecen sin cambios, la señal *Listo* se encuentra desactivada. En el estado S_7 se realiza el primer producto parcial, el registro del acumulador se encuentra cargando datos, los registros de resultados permanecen sin cambios, la señal *Listo* se encuentra desactivada. En el estado S_8 se realiza el segundo producto parcial, el registro del acumulador se encuentra cargando datos, los registros de resultados permanecen sin cambios, la señal *Listo* se encuentra desactivada. En el estado S_9 se realiza el tercer producto parcial, el registro del acumulador se borra, el registro3 presenta el resultado *S3*, los demás registros de resultados permanecen sin cambios, la señal *Listo* se encuentra desactivada.

Cuando carga el ultimo resultado la maquina pasa al estado S_{10} donde se espera hasta que la señal *Listo* se haya desactivado para regresar al estado inicial S_0, las condiciones de control son las mismas que el estado inicial en la figura 3.9b se muestra el grafo de la máquina de estados.

4) Se utiliza la función de activación, para calcular la salida de la neurona de la capa escondida, se denomina bloque lógico de la señal de activación, para la descripción de la señal de activación, esta se implementara con la aproximación de funciones por medio de polinomios, Para el cálculo eficiente de los valores de un polinomio se utiliza el algoritmo de Horner, también conocido como multiplicación anidada y como división sintética. La regla de Horner para la evaluación de polinomios establece que una expresión polinómica puede rescribirse como una factorización anidada. Por ejemplo, el siguiente polinomio de grado 5:

$$p(x) = a_5x^5 + a_4x^4 + a_3x^3 + a_2x^2 + a_1x + a_0 \qquad (3.3)$$

puede rescribirse como:

$$p(x) = ((((a_5x + a_4)x + a_3)x + a_2)x + a_1)x + a_0 \qquad (3.4)$$

Esta aproximación se realizo en MatLab dándonos como resultado un polinomio de cuarto orden, la ecuación (3.3) muestra la ecuación de la función de activación.

$$h_j = \frac{1}{1 + e^{-S_j}} \qquad (3.5)$$

Donde j = 1,2,3:

Desarrollando la ecuación (3.3) hasta j = 3 obtenemos las siguientes ecuaciones:

$$h_1 = \frac{1}{1+e^{-S_1}}$$
$$h_2 = \frac{1}{1+e^{-S_2}}$$
$$h_3 = \frac{1}{1+e^{-S_3}}$$

La ecuación de aproximación de funciones por polinomios, utilizando el algoritmo de Horner de la función de activación, podemos rescribir las ecuaciones anteriores como se muestra en las siguientes ecuaciones:

$$h1 = \left[(0+a_4)S1 + a_3\}S1 + a_2\}S1 + a_1\}S1 + a_0\right]$$
$$h2 = \left[(0+a_4)S2 + a_3\}S2 + a_2\}S2 + a_1\}S2 + a_0\right]$$
$$h3 = \left[(0+a_4)S3 + a_3\}S3 + a_2\}S3 + a_1\}S3 + a_0\right]$$

En la figura 3.12 mostramos el bloque general para la representación de las ecuaciones anteriores en forma digital.

Figura 3.12.- Bloque lógico de la señal de activación.

De acuerdo con las ecuaciones de aproximación por polinomios la estructura interna del Bloque lógico de la figura 3.12. Se muestra en la figura 3.13.

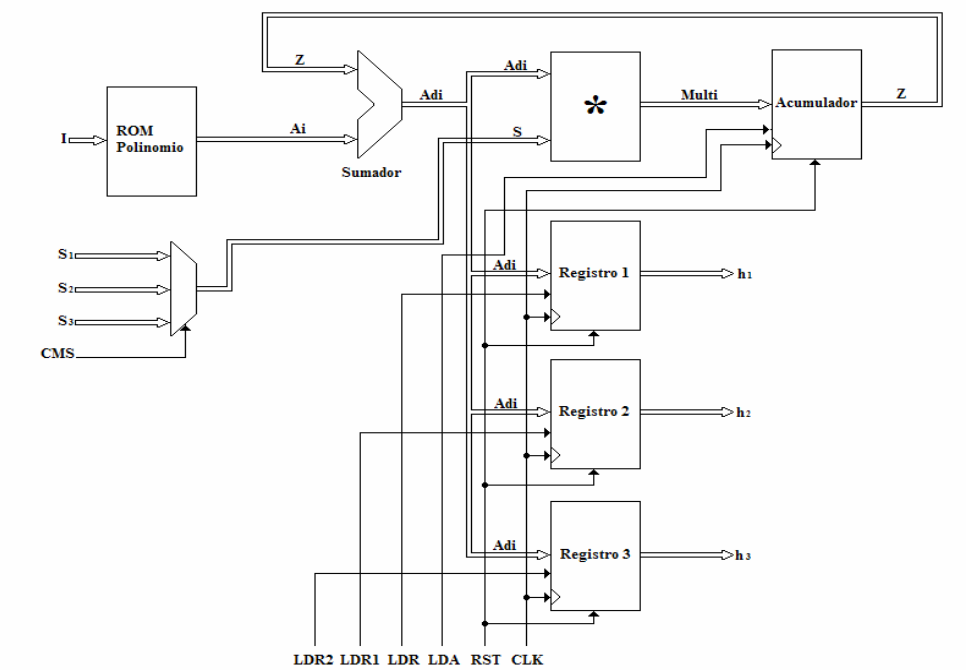


Figura 3.13.- Estructura interna del bloque lógico de la señal de activación.

La estructura digital de la figura 3.13 esta compuesta por una memoria ROM que contiene los valores de los coeficientes de las ecuaciones del resultado de la aproximación, los cuales constan de 18 bits, un multiplexor de 4 a 1 que contiene las entradas *S1*, *S2* y *S3*, cada una de 18 bits, un sumador, un bloque multiplicador, un acumulador y tres registros de 18 bits cada uno para los resultados. En la figura 3.14 se muestra el bloque del control de las operaciones que se realizan para cada entrada, así como en la figura 3.115 su diagrama de Estados.

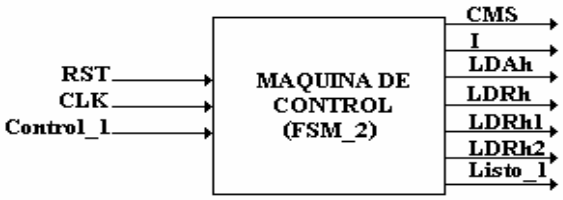


Figura 3.14.- Bloque general de la maquina de control de la señal de activación

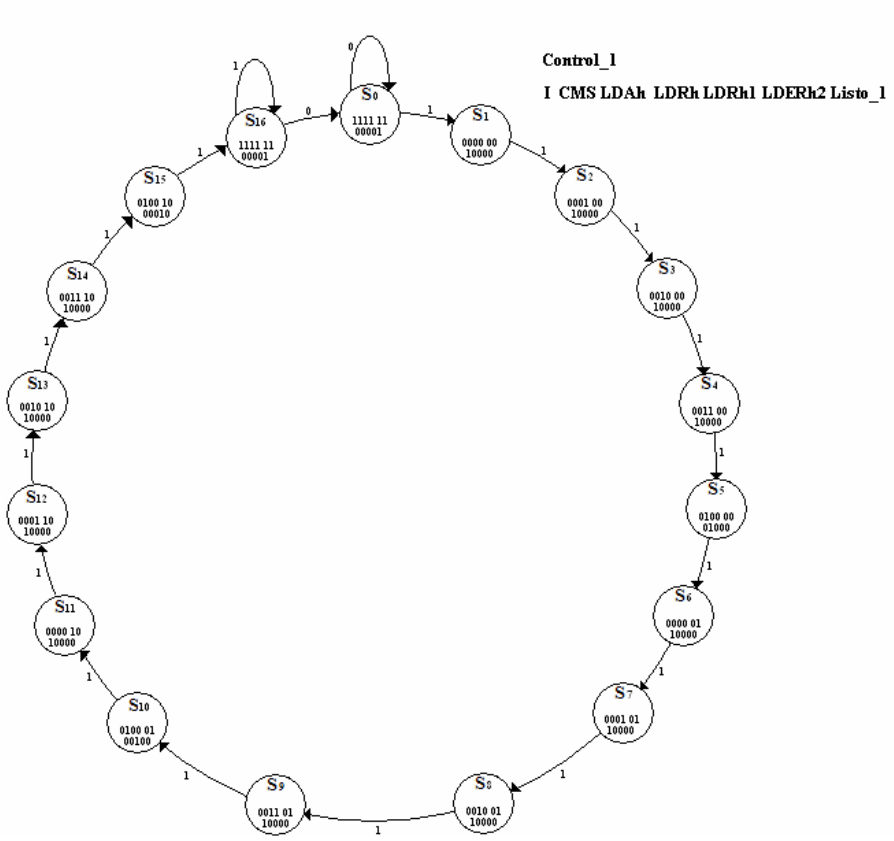


Figura 3.15.- Grafo o máquina de control de la señal de activación

La maquina de control de la figura 3.15 empieza en un estado S_0 donde la señal de *Control_1* esta desactivada, la señal *Listo_1* esta activada, los multiplexores su salida es cero, el acumulador permanece sin cambios, así como los registros de resultados. Cuando la señal de *Control_1* se activa, del estado S_0 pasa al estado S_1, se selecciona el primer *índice* cero de la memoria ROM del primer coeficiente de las ecuaciones resultantes de la aproximación por polinomios, así como *S1*, se realiza la primera operación guardándose en el acumulador, los registros de resultados permanecen sin cambios. La señal de *Control*_1 se vuelve a activar, la máquina de control pasa del estado S_1 al S_2, realizándose otra operación, seleccionando el índice uno y la señal de entrada *S1*, (se realiza la operación de suma y multiplicación de acuerdo a la figura 311) el acumulador permanece activo), los registros de resultados permanecen sin cambios, la señal *Listo_1* se encuentra desactivada. Al activarse la señal de *Control_1* la máquina de estados

pasa del Estado S_2 al S_3 se selecciona el *índice* dos y la señal de entrada *S1* efectuándose la operación correspondiente, el acumulador se encuentra activado, los registros de resultados permanecen sin cambios, la señal de *Listo_1* se encuentra desactivada. La señal de *Control_1* se activa por lo tanto se pasa del estado S_3 al S_4 se selecciona el índice 3 y la señal de entrada *S1* efectuándose la operación correspondiente, el acumulador esta activado. Los registros de resultado permanecen sin cambios, la señal de Listo_1 se encuentra desactivada. Cuando ocurre otra activación de la señal de *Control_1*, la máquina de estados pasa del estado S_4 al S_5 se selecciona el índice cinco de la memoria ROM y la señal *S1* efectuándose la operación, el acumulador se desactiva, el registro1 presenta el resultado de *h1*, los demás registros permanecen sin cambios, la señal listo_1 esta desactivada. Todo este proceso se repite de la misma manera para las señales de entrada *S2* y *S3*, hasta que se llega al estado S_{16} donde la señal de *Control_1* se desactiva, la salida de los multiplexores es cero, no se realiza ninguna operación, la señal *listo_1* se activa, esperando más datos para empezar de nuevo el proceso.

5) Se calcula *r* que es la multiplicación de la salida de la neurona de la capa escondida por los peso *Vj*, se denominara bloque lógico de multiplicación de los parámetros de salida de neurona de la capa escondida con los pesos *Vj*

$$r = \sum_{j=1}^{3} v_j \; h_j \tag{3.6}$$

Desarrollando la ecuación (3.6) obtenemos:

$$r = h_1 V_1 + h_2 V_2 + h_3 V_3 \tag{3.7}$$

La representación digital de la ecuación (3.6) de este paso del algoritmo de aprendizaje se muestra en la figura 3.16.

Figura 3.16.- Bloque lógico de multiplicación de los parámetros de salida de neurona de la capa escondida con los pesos Vj.

La representación digital de la estructura interna de este bloque lógico que corresponde a la ecuación (3.7) se muestra en la figura 3.17.

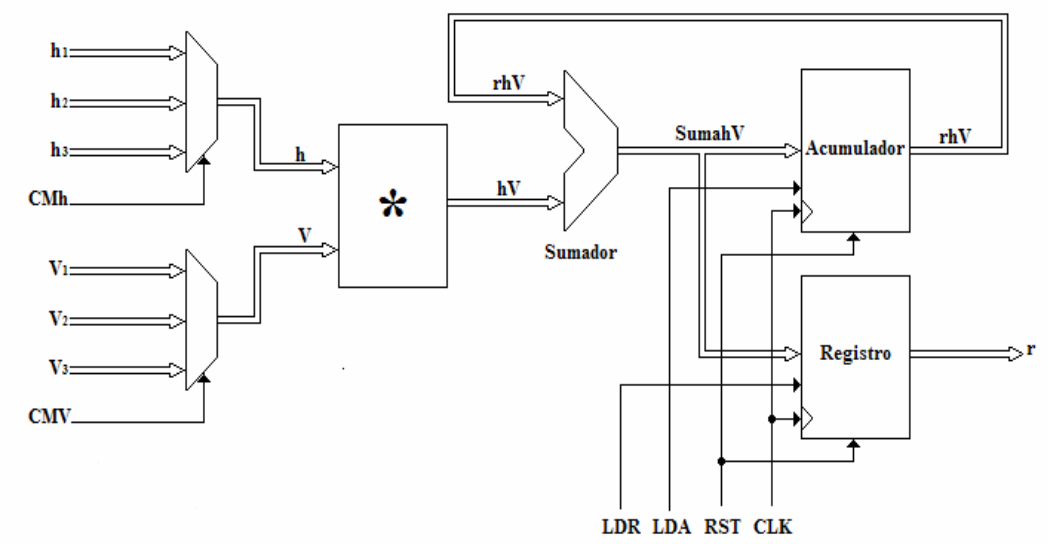


Figura 3.17.- Estructura interna del Bloque lógico de multiplicación de los parámetros de salida de neurona de la capa escondida con los pesos Vj.

La estructura interna del bloque lógico de la figura 3.14 esta compuesta por dos multiplexores de 4 a 1 para las señales de entrada de 18 bits *h1*,*h2*,*h3*, *V1*, *V2*, y

V3, por un multiplicador de 18 bits, un sumador de 18 bits, un acumulador de 18 bits y un registro de 18 bits para presentar el resultado, el bloque de la maquina de control y su diagrama de estados para la realización de las operaciones del desarrollo de la ecuación (3.7) y su diagrama de estados de la máquina de control, se muestran en las figuras 3.18 y 3.19.

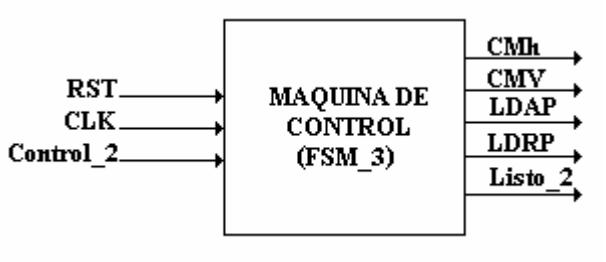


Figura 3.18.- Bloque general de la maquina de control para el calculo de r.

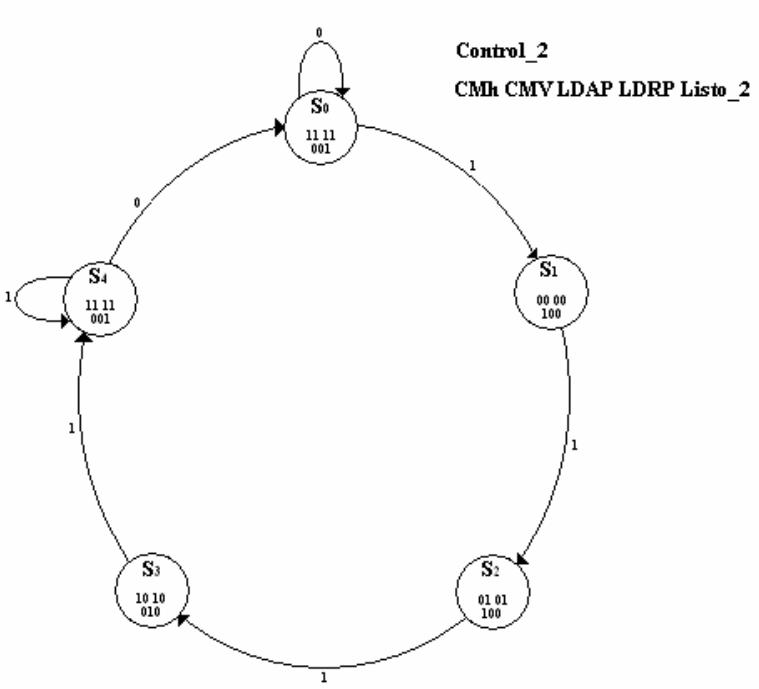


Figura 3.19.- Grafo de la máquina de control para la realización de las operaciones.

La maquina de control de la figura 3.19 empieza con un estado S_0 en donde la señal *Listo_2* se encuentra activada, la salida de los multiplexores es cero, el acumulador y el registro permanecen sin cambios. Cuando se activa la señal de *Control_2*, la máquina de control pasa del estado S_0 al S_1, los multiplexores tienen a la salida las señales *h1* y *V1* para realizar la multiplicación de estas, el acumulador se activa guardando la primera operación, el registro permanece sin cambios y la señal *Listo_2* se desactiva, En la siguiente transición positiva del reloj maestro se activa la señal de *Control_2*, la máquina de control pasa del estado S_1 al S_2, activando la salida de los multiplexores *h2* y *V2* para la segunda multiplicación, ahora sumándose con la primera, y guardándose en el acumulador que sigue activo, el registro y la señal *Listo_2* permanecen sin cambios. La señal de *Control_2* se activa por lo tanto la máquina de control pasa del estado S_2 al S_3, los multiplexores tienen a la salida las señales *h3* y *V3* para realizar la multiplicación de estas, la cual se suma con las dos anteriores que se guardaron en el acumulador, en donde se desactiva, el registro se activa y presenta el resultado *r*, la señal *Listo_2* esta desactivada, la señal de control se activa, la máquina de control pasa del estado S_3 al S_4, donde los multiplexores presentan en las salidas cero, el acumulador y el registro se desactivan, la señal *Listo_2* se activa, para los demás estados, se regresan al estado inicial.

6) Se utiliza la función de activación, para calcular la salida de la neurona de la capa de salida, se denomina bloque lógico de la señal de activación I (Para el calculo de esta función se realizará por medio de aproximaciones por polinomios)

$$u(t) = \frac{1}{1+e^{-r}} \tag{3.8}$$

La ecuación de aproximación de funciones por polinomios por medio del algoritmo de Horner de la función de activación se muestra en la siguiente ecuación:

$$u = \left[(0 + a_4)r + a_3\}r + a_2\}r + a_1\}r + a_0\right] \tag{3.9}$$

En la figura 3.20 mostramos el bloque general para la representación de la ecuación anterior en forma digital.

Figura 3.20.- Bloque lógico de la señal de activación.

La aproximación por polinomios de la ecuación (3.9) obtenemos la estructura interna del bloque lógico anterior se muestra en la figura 3.21

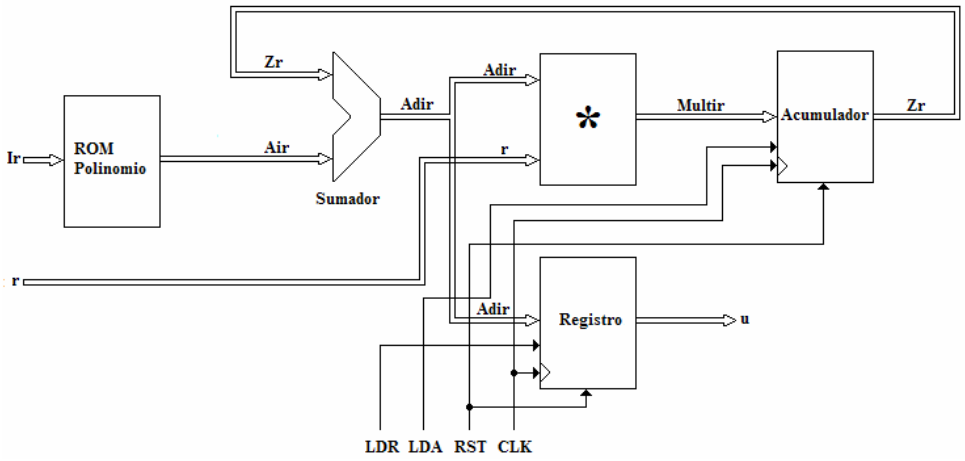


Figura 3.21.- Estructura interna del Bloque lógico de la señal de activación.

La figura 3.21 consta de una memoria ROM que contiene los coeficientes de la ecuación de los polinomios, los cuales son de 18 bits, un sumador, multiplicador, acumulador y un registro todos para señales de 18 bits. El bloque de controla y su diagrama de estados para el control de las operaciones se muestra en la figura 22 y 23 respectivamente.

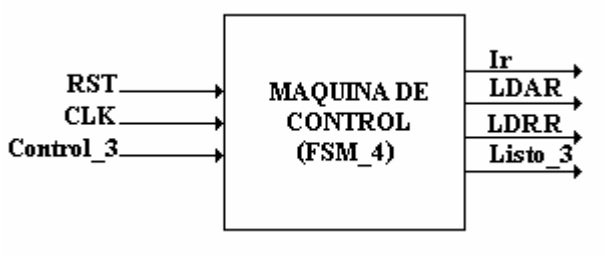


Figura 3.22.- Bloque general de la maquina de control

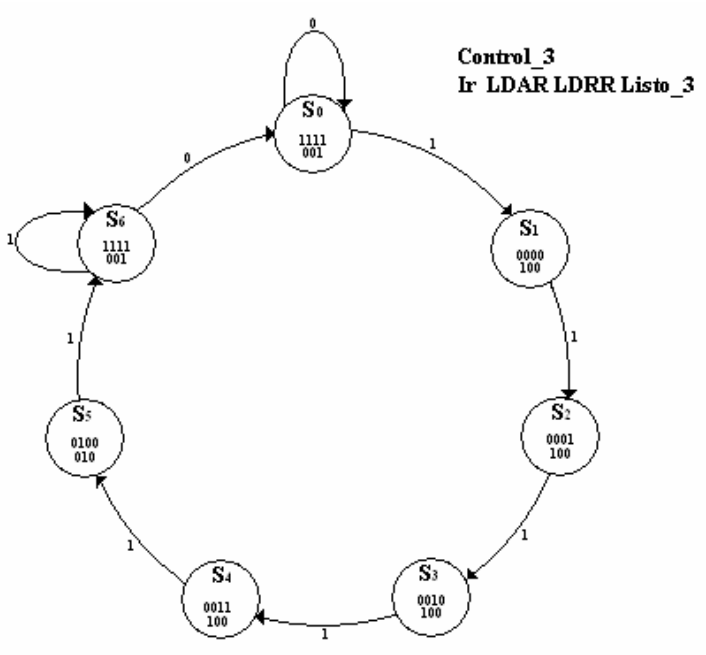


Figura 3.23.- Grafo de la máquina de control.

La máquina de control de la figura 3.23 empieza con un estado inicial S_0, donde la memoria ROM a su salida tiene un cero, estando desactivada, el acumulador y el registro están desactivados, la señal de *Listo_3* esta activada. La señal de *Control_3* se activa la maquina de control pasa del estado S_0 al S_1, donde se activa la memoria ROM presentando a la salida el primer *índice*, para multiplicarse con la señal de entrada *r*, el acumulador se activa para guardar el dato, el registro permanece sin cambios y la señal de *Listo_3* se desactiva. Cuando la señal de *Control_3* es activada nuevamente la máquina de control pasa

del estado S_1 al S_2 seleccionándose el segundo índice para la multiplicación con la señal *r,* y así realizándose la suma con anterior operación, guardándole este resultado en el acumulador que esta activado, el registro permanece sin cambos, la señal *Listo_3* se encuentra desactivada. Al activarse la señal de *Control_3* la maquina de control pasa del estado S_2 al S_3, el tercer *índice* se multiplica con la señal de entrada *r*, sumándose con los datos que hay en el acumulador que esta activado, el registro permanece sin cambios y la señal de *Listo_3* se encuentra desactivada. La señal de *Control_3* se activa el control pasa al estado S_4, el cuarto índice se multiplica con al señal de entrada *r*, sumándose a los datos que están en el acumulador que esta activado, el registro permanece sin cambios. Se activa la señal de *Control_3* la máquina pasa al estado S_5, el quinto *índice* se multiplica con *r* que se suma con las operaciones anteriores, presentado el registro el resultado u, la señal de *Listo_3* se encuentra desactivada, al activarse la señal de *Control_3* se pasa al estado S_6 donde se terminan las operaciones, por lo tanto el acumulador y el registro se desactivan, y la señal de *Listo_3* se activa, La señal de *Control_3* de desactiva y pasa al estado inicial S_0 para empezar de nuevo el proceso.

7) En este paso que es el último del algoritmo de aprendizaje de retropropagación o propagación hacia atrás, es la actualización de los pesos Wij y Vj de la red neuronal autoajustable, donde denominaremos Bloque Lógico de actualización de pesos de Wij, y Bloque Lógico de actualización de pesos de Vj, respectivamente, los cuales se muestran en la figura 3.24 y figura 3.28

$$v_j(t+1) = v_j(t) + \eta\delta_1 h_j \tag{3.10}$$

$$w_{ji}(t+1) = w_{ji}(t) + \eta\delta_2 x_i \tag{3.11}$$

Donde:

$$\delta_1 = (e_y)\left[u(1-u)\right]$$

$$\delta_2 = \delta_1 v_j h_j (1-h_j)$$

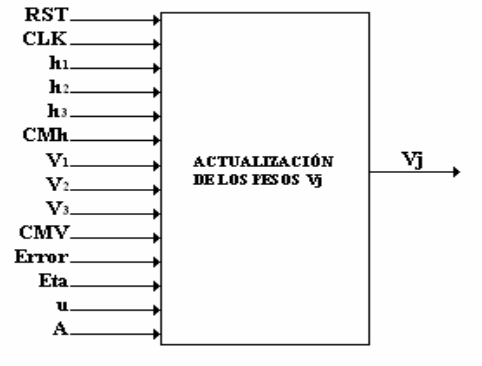


Figura 3.24.- Bloque lógico de actualización de los pesos Vj .

La representación digital de la ecuación (3.10), es la estructura interna del bloque lógico de actualización de los pesos Vj, la cual consta de tres multiplicadores de 18 bits, un sumador, un restador, un mutliplexor 4 a 1 y tres registros de 18 bits como se muestra en la figura 3.25.

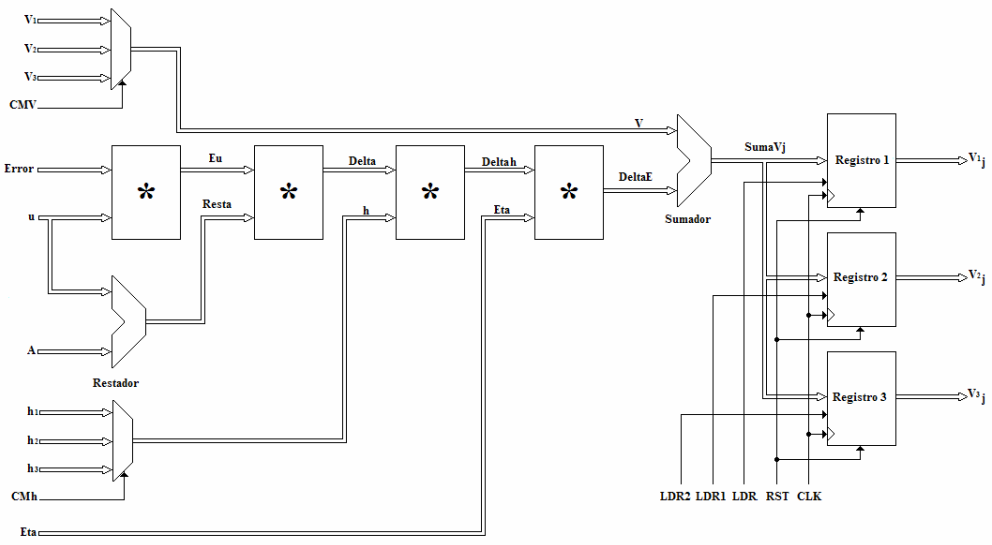


Figura 3.25.- Estructura interna del Bloque lógico de actualización de los pesos Vj.

La máquina de control para la realización de las operaciones de la ecuación (3.10) representado en forma digital en la figura 3.25, se muestra en la figura 3.26.

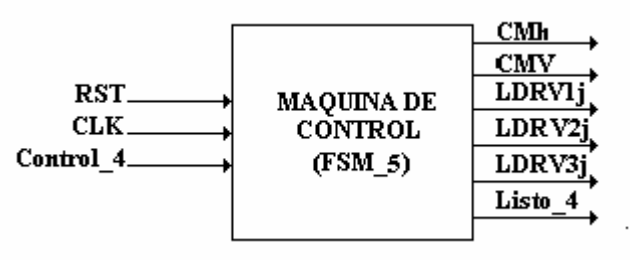


Figura 3.26.- Bloque general de la maquina de control

El grafo de la máquina de control, la cual nos ayuda en el control de la operaciones que se van a realizar en este bloque se muestra en la figura 3.27.

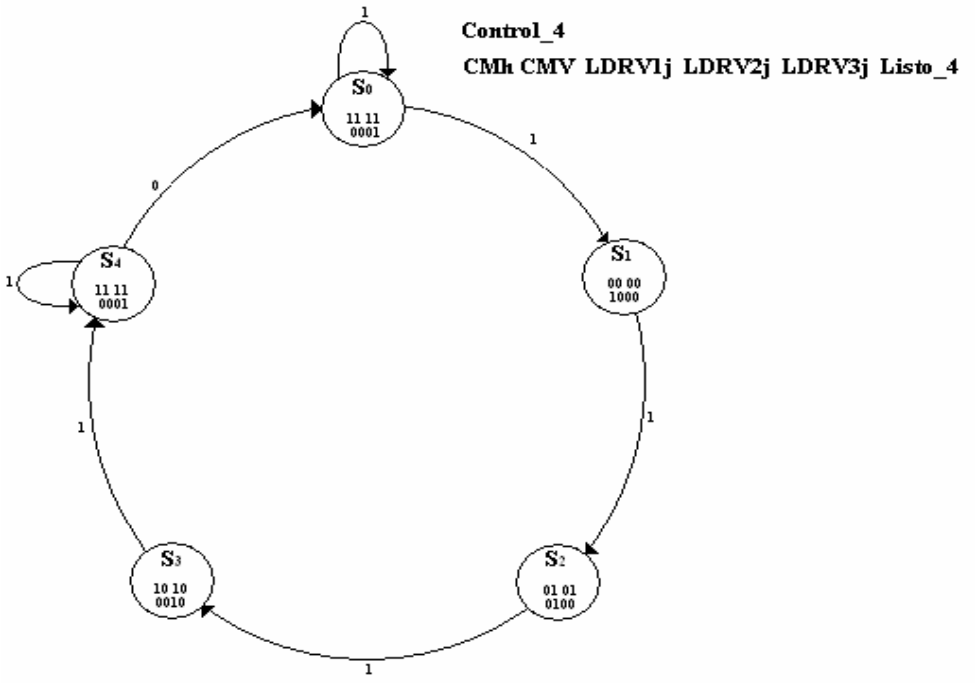


Figura 3.27.- Grafo de máquina de control para la realización de las operaciones.

La máquina de control de las operaciones del bloque lógico de actualización de pesos *Vj*, e la figuitra 3.27 comienza en un estado inicial S_0 donde las salidas de los multiplexores están en cero, los registros de resultados permanecen sin cambios y la señal de *Listo_4* esta activada. La señal de *Control_4* cuando se activa la máquina de control pasa al estado S_1, habilitándose la señal de entrada *h1*

y *V1* para realizar la operación descrita en la ecuación (3.10), el registro 1 se activa presentando el resultado de *V1j*, los demás registros permanecen sin cambios, la señal de Listo_4 esta desactivada. La señal de *Control_4* se activa nuevamente la maquina de control pasa al estado S_2, se habilitan las señales de entrada *h2* y *V2*, realizándose la operación correspondiente, el registro 1 se deshabilita, el registro 2 se activa mostrando el resultado *V2j*, el tercer registro permanece sin cambios, la señal de *Listo_4* esta desactivada. La señal de *Control_4* se activa, el control pasa al estado S_3, seleccionándose las señales de entrada *h3* y *V3*, para la operación requerida, el registro 1 permanece sin cambios, el registro 2 se desactiva, el registro 3 se activa para presentar el resultado *V3j*, la señal de *Listo_4* esta desactivada. La señal de *Control_4* se activa, el control pasa al estado S_4, donde los multiplexores en la salida presentan un cero, los registros se desactivan y la señal de *Listo_4* se activa indicando que el proceso de las operaciones ha terminado hasta que la señal de *Control_4* se active nuevamente para realizar otra vez el proceso.

Ahora se muestra el bloque lógico para la actualización de los pesos Wji.

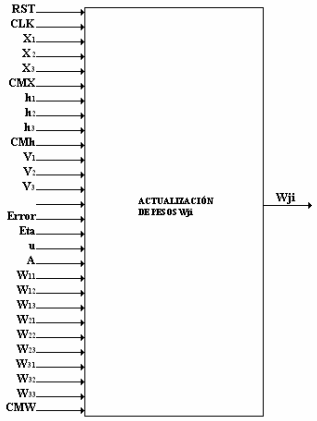


Figura 3.28.- Bloque lógico de actualización de los pesos Wji.

En la figura 3.29 se muestra la estructura interna en forma digital de la ecuación de la actualización de los pesos Wji. La cual esta representada por la ecuación (3.11)

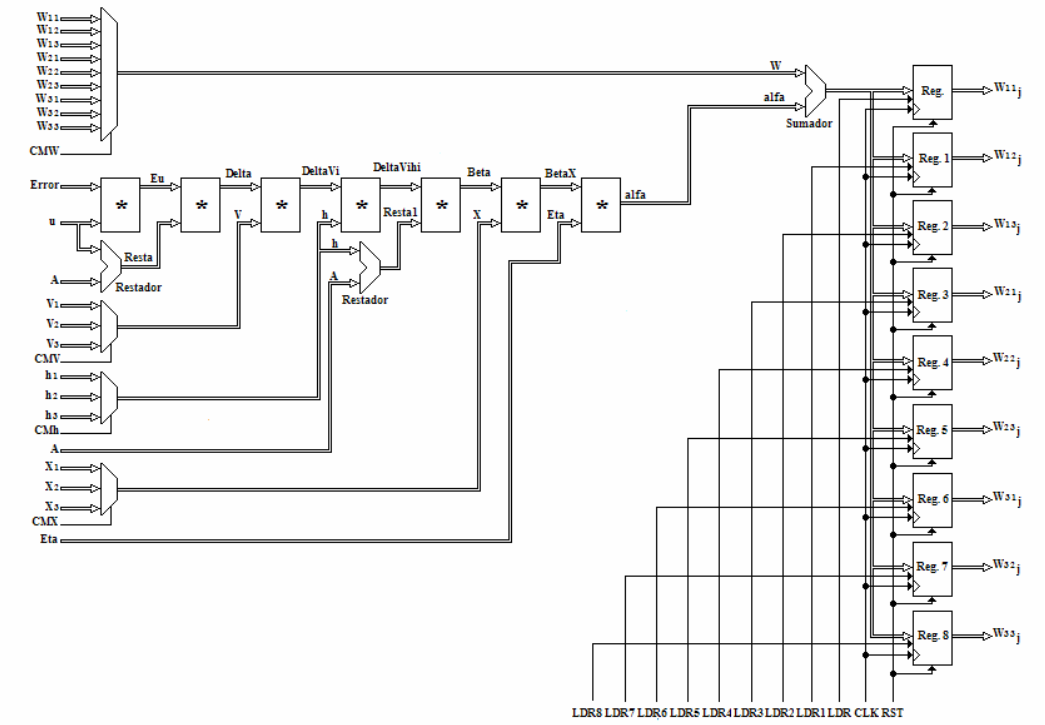


Figura 3.29.- Estructura interna del Bloque lógico de actualización de los pesos Wji.

La estructura interna de la figura 3.29 esta formada por tres multiplexores de 4 a 1, un multiplexor de 12 a 1, un restador, un sumador y nueve registros de resultados, todos ellos de 18 bits. Para la realización de las operaciones en la figura 3.30 se muestra su bloque lógico de la máquina de control, y en la figura 3.31 se muestra su diagrama de estados.

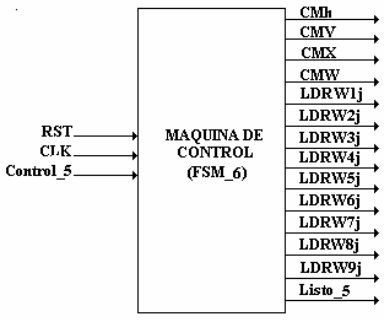


Figura 3.30.- Bloque general de la maquina de control Wji.

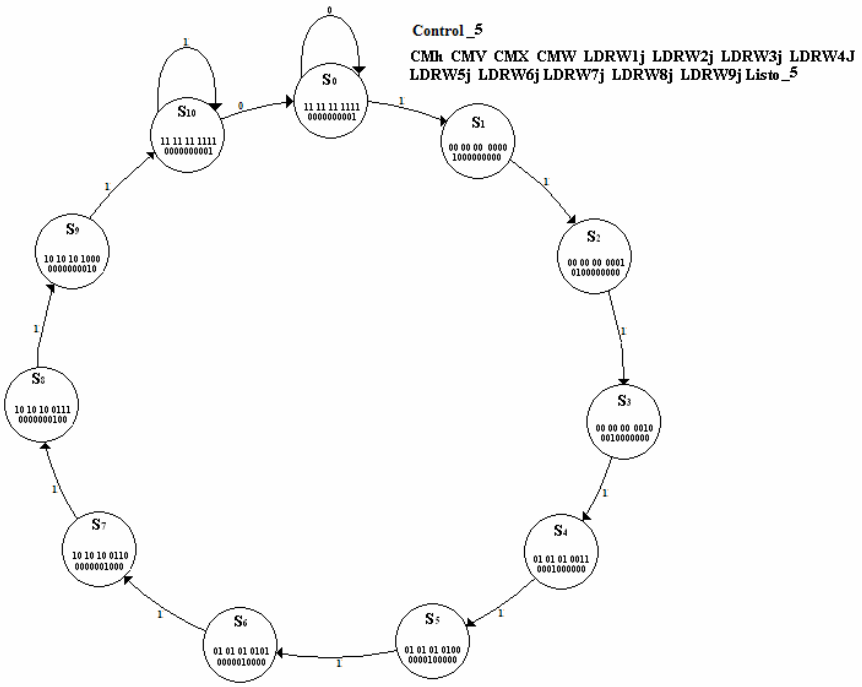


Figura 3.31.- Grafo de la máquina de control para Wji.

La máquina de control de las operaciones del bloque lógico de actualización de pesos *Wij*, comienza en un estado inicial S_0 donde las salidas de los multiplexores están en cero, los registros de resultados permanecen sin cambios y la señal de *Listo_5* esta activada. Al activarse la señal de *Control_5*, la máquina de control pasa al estado S_1, los multiplexores presentan las señales de entrada *h1*, *V1*, *X1* y *W11*, se realizan las operaciones indicadas en la ecuación (3.11), el registro se activa, dando el resultado de *W11j*, los demás registros permanecen sin cambios, la señal de *Listo_5* se desactiva, La señal de *Control_5* se activa nuevamente la máquina de control pasa al estado S_2, los multiplexores seleccionan la entradas correspondientes a *h1*, *V1*, *X1* y *W12*, realizando la s operaciones correspondientes, el registro se desactiva, el registro1 se activa presentando el resultado *W12j*, los demás registros permanecen sin cambios, la señal de *Listo_5* esta desactivada, este proceso se repite hasta que la maquina de control llega al estado S_9 donde los mutiplexores tienen a la salida las señales de entrada *h3*, *V3*; *X3* y *W33*, se realizan las operaciones correspondientes, el registro8 se activa,

dando el resultado *W33j*, mientras que el registro7 se desactiva, los demás registros permanecen sin cambios, la señal de *Listo_5* esta desactivada, la señal de *Control_5* se activa la máquina de control pasa al estado S_{10}, donde los multiplexores presentan a la salida cero, no se realiza ninguna operación, los registros se desactivan, la señal de *Listo_5* se activa, cuando la señal de *Control_5* se desactiva la máquina de control pasa al estado inicial S_0, esperando que la señal de *Control_5* se active para empezar de nuevo el proceso.

En el apartado anterior se realizo la descripción de los pasos del algoritmo de aprendizaje de Retropropagación o Propagación hacia atrás, procedemos a unir todos los bloques de lógicos de los pasos para obtener el bloque general de la Red Neuronal Artificial Autoajustable, como se muestra en la figura 3.32.

Figura 3.32.- Bloque Lógico de la Red Neuronal Artificial Autoajustable.

La estructura interna de la figura del Bloque General de la Red Neuronal Artificial Autoajustable en la figura 3.33 se muestra como están conectados todos los bloques lógicos de los 7 pasos del algoritmo de aprendizaje de Retropropagación.

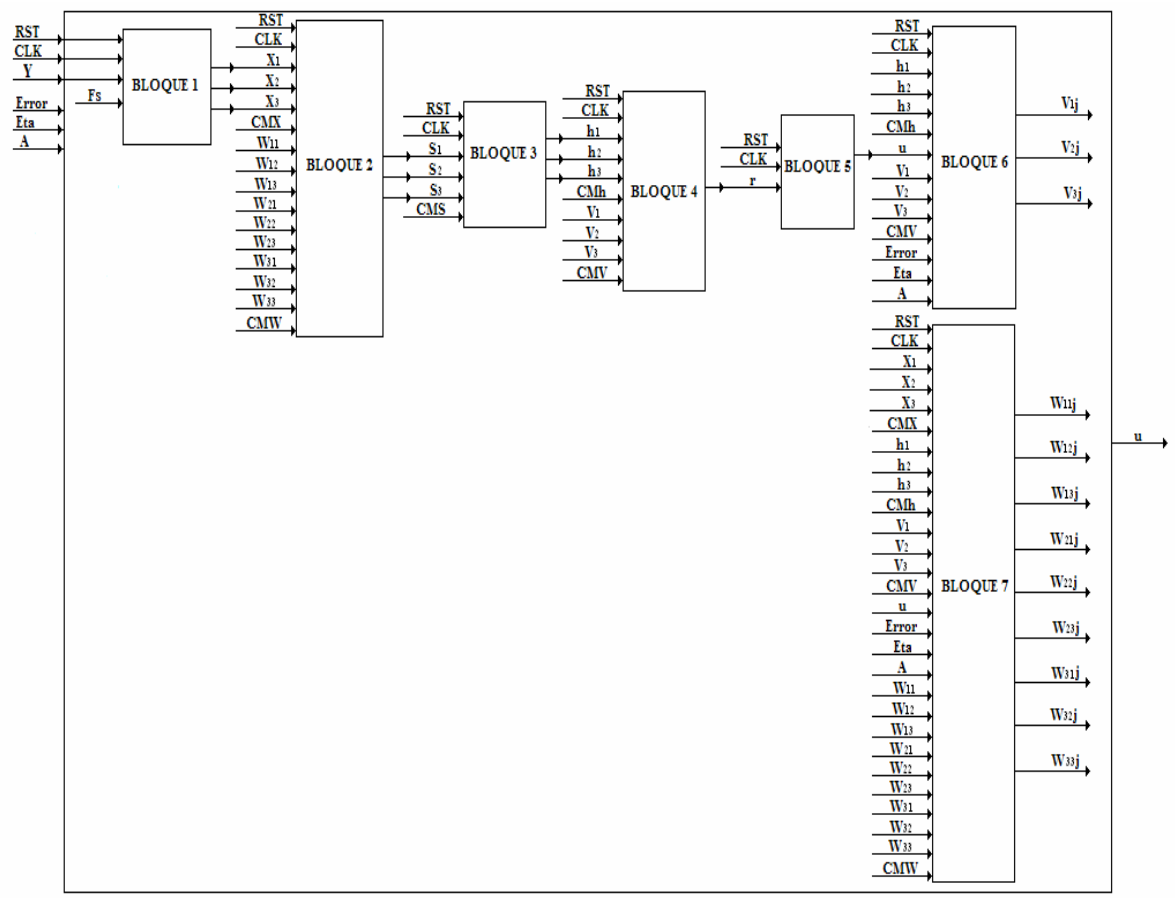


Figura 3.33 Estructura Interna del bloque Lógico de la red neuronal artificial Autoajustable.

Para el control de las operaciones realizadas en cada bloque lógico, es necesario tener una máquina de control, para sincronizar todas las máquinas de control de cada bloque lógico para darle el tiempo suficiente de que cada bloque lógico realice su operación, para que cada máquina de control termine su proceso, en la figura 3.34 se muestra el bloque general de la máquina de control con sus entradas y sus salidas.

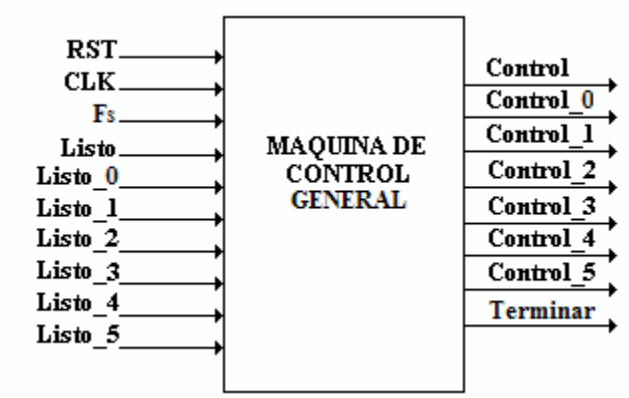


Figura 3.34.- Bloque General de la máquina de Control de la RNAA.

A continuación se muestra en la figura 3.35, el grafo del bloque general de la máquina de control de la Red Neuronal Artificial Autoajustable.

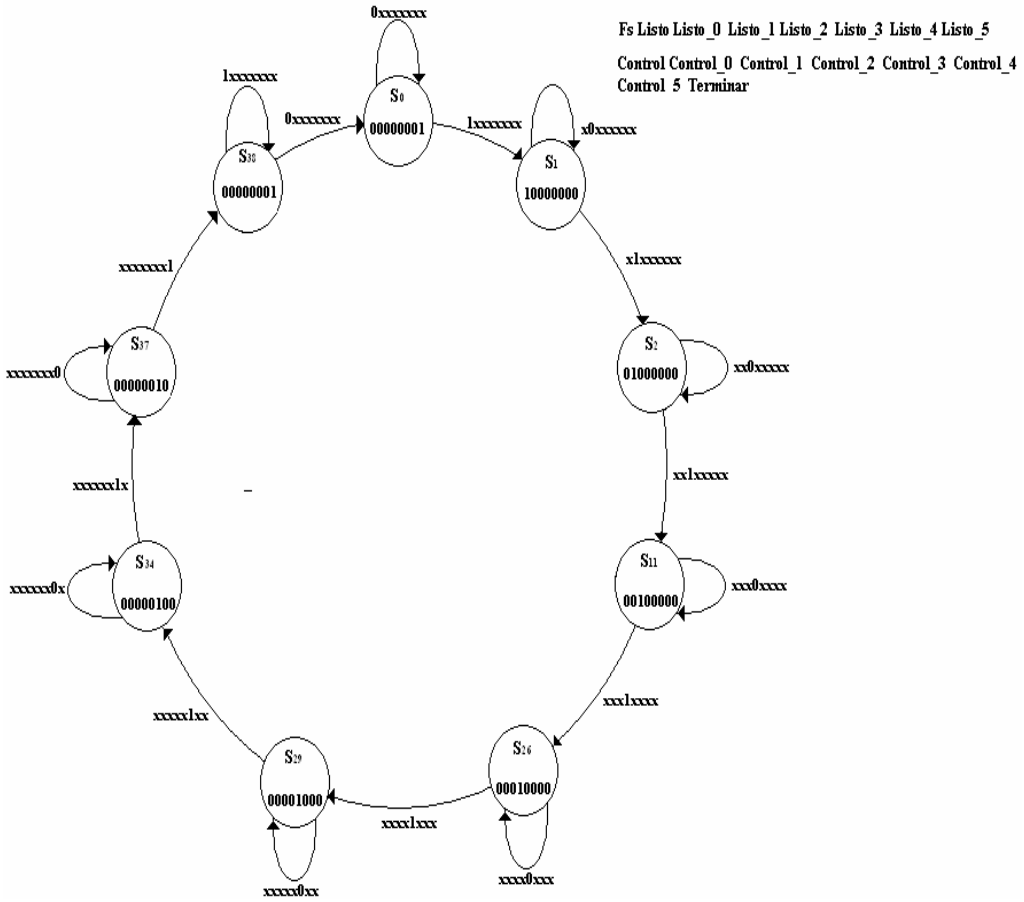


Figura 3.35.- Grafo de la máquina de control de la RNAA.

Se observa de la figura 3.28 que la máquina de control del bloque general de la red neuronal artificial Autoajustable, con esta máquina sincronizamos las demás maquinas de control de los bloques individuales, este control empieza en un estado S_0 donde la señal *Fs* es la frecuencia de muestreo de un milisegundo, esta se encuentra desactivada, las señales de *Control, Control_0, Control_1, Control_2, Control_3, Control_4, Control_5*, se encuentran desactivadas, la señal *Terminar* se encuentra activada, mientras las señales *Listo, Listo_0, Listo_1, Listo_2, Listo_3, Listo_4, Listo_5*, se encuentran desactivadas. Cuando la señal *Fs* se activa la maquina de control pasa del estado S_0 al S_1, la señal *Listo* se activa, la primera máquina de control empieza a realizar su operación correspondiente, la señales *Listo_0, Listo_1, Listo_2, Listo_3, Listo_4, Listo_5*, no importa que valor tengan, la señal *Control* se activa iniciando el corrimiento de la señal de referencia. Las señales *Control_0, Control_1, Control_2, Control_3, Control_4, Control_5*, *Terminar* se encuentran desactivadas, con la siguiente transición positiva de reloj, la máquina de control pasa del estado S_1 al S_2, la señal *Listo_0* se activa, empieza a trabajar la segunda máquina de control, con las operaciones correspondientes, la señal *Listo* se desactiva, y la señal *Listo_0* se activa, mientras las señales *Listo, Listo_1, Listo_2, Listo_3, Listo_4, Listo_5*, no importa si están desactivas o activadas, la señal de *Control* se desactiva, y la señal de *Control_0* se activa, las señales de *Control_1, Control_2, Control_3, Control_4, Control_5* y *Terminar* se encuentran desactivadas. En esta segunda máquina de control tenemos que esperar a que se realicen las operaciones, para que se realice esto es necesario esperar del estado S_3 al $S_{10,}$ de esta forma podemos sincronizar las máquinas de control, la señal de *Control_0* se desactiva, mientras la señal de *Terminar* se activa, las señales *Listo, Listo_0,Listo_1, Listo_2, Listo_3, Listo_4, Listo_5*, no importa que valor tomen, las señales *Control*, *Control_0, Control_1, Control_2, Control_3, Control_4, Control_5*, se encuentran todas desactivadas mientras se realizan las operaciones correspondientes a esta máquina. Cuando la máquina de control pasa al estado $S_{11,}$ termina sus operaciones la máquina anterior, por lo siguiente se activa la señal de

Listo_1 por que la señal de *Control_1* es activada y la señal de *Terminar* se desactiva, la tercera máquina de control empieza a realizar las operaciones correspondientes con los resultados de la operaciones anteriores, una vez que comienza a funcionar, las señales se encuentran desactivadas como en el proceso anterior, que tenemos que esperar a que realice sus operaciones, para que no haya errores, la señal de Terminar se encuentra activada, esto se realiza en los estados S_{12} al S_{25}. La máquina de control pasa al estado al estado S_{26}, la señal de *Listo_2* es activada, por lo tanto la señal de *Control_2* se activa y la señal de Terminar se desactiva, empieza a trabajar la cuarta maquina de control, en este estado las *Listo, Listo_0, Listo_1, Listo_3, Listo_4, Listo_5*, no importa que valor tomen, las señales *Control*, *Control_0, Control_1, Control_3, Control_4, Control_5*, se encuentran todas desactivadas mientras se realizan las operaciones correspondientes a esta máquina. En los estados del S_{27} al S_{28}. Las señales *Listo, Listo_0,Listo_1, Listo_2, Listo_3, Listo_4, Listo_5*, no importa que valor tomen, las señales *Control*, *Control_0, Control_1, Control_2, Control_3, Control_4, Control_5*, se encuentran todas desactivadas mientras se realizan las operaciones y la señal *Terminar* se encuentra activada. La máquina de control pasa al estado al estado S_{29}, la señal de *Listo_3* es activada, por lo tanto la señal de *Control_3* se activa y la señal de Terminar se desactiva, empieza a trabajar la quinta maquina de control, en este estado las *Listo, Listo_0, Listo_1, Listo_2, Listo_4, Listo_5*, no importa que valor tomen, las señales *Control*, *Control_0, Control_1, Control_2, Control_4, Control_5*, se encuentran todas desactivadas mientras se realizan las operaciones correspondientes a esta máquina. En los estados del S_{30} al S_{33}. Las señales *Listo, Listo_0, Listo_1, Listo_2, Listo_3, Listo_4, Listo_5*, no importa que valor tomen, las señales *Control*, *Control_0, Control_1, Control_2, Control_3, Control_4, Control_5*, se encuentran todas desactivadas mientras se realizan las operaciones y la señal *Terminar* se encuentra activada. La máquina de control pasa al estado al estado S_{34}, la señal de *Listo_4* es activada, por lo tanto la señal de *Control_4* se activa y la señal de Terminar se desactiva, empieza a trabajar la sexta maquina de control, en este estado las *Listo, Listo_0, Listo_1, Listo_2,*

Listo_3, Listo_5, no importa que valor tomen, las señales *Control*, *Control_0, Control_1, Control_2, Control_3, Control_5*, se encuentran todas desactivadas mientras se realizan las operaciones correspondientes a esta máquina. En los estados del S_{35} al S_{36}. Las señales *Listo, Listo_0, Listo_1, Listo_2, Listo_3, Listo_4, Listo_5*, no importa que valor tomen, las señales *Control*, *Control_0, Control_1, Control_2, Control_3, Control_4, Control_5*, se encuentran todas desactivadas mientras se realizan las operaciones y la señal *Terminar* se encuentra activada. La máquina de control pasa al estado al estado S_{37}, la señal de *Listo_5* es activada, por lo tanto la señal de *Control_5* se activa y la señal de Terminar se desactiva, empieza a trabajar la séptima máquina de control, en este estado las *Listo, Listo_0, Listo_1, Listo_2, Listo_3, Listo_4*, no importa que valor tomen, las señales *Control*, *Control_0, Control_1, Control_2, Control_3, Control_4*, se encuentran todas desactivadas mientras se realizan las operaciones correspondientes a esta máquina. En los estados del S_{38} a todos los restantes. Las señales *Listo, Listo_0, Listo_1, Listo_2, Listo_3, Listo_4, Listo_5*, no importa que valor tomen, las señales *Control*, *Control_0, Control_1, Control_2, Control_3, Control_4, Control_5*, se encuentran todas desactivadas mientras se realizan las operaciones y la señal *Terminar* se encuentra activada.

3.5 Conclusiones

La representación digital de las ecuaciones que contiene el algoritmo de aprendizaje de retropropagación de la red neuronal artificial autoajustable, modelada en un FPGA fue implementada satisfactoriamente, lo cual con este estudio nos permite realizar investigaciones a futuro del procesamiento de señales en hardware.

La simulación del algoritmo de aprendizaje de retropropagación de la red neuronal artificial autoajustable, nos permite conocer las tecnologías de diseño, para migrar al modelado de sistemas inteligentes para el control de sistemas muy complejos, en circuitos programables FPGA.

CAPÍTULO IV.
RESULTADOS Y CONCLUSIONES.

4.1.-Introducción.

Comprobando y obteniendo la exactitud, el intervalo y la precisión de los datos, en un formato determinado de 2:16 para la representación de los números de punto fijo, para determinar la posición del punto decimal en el resultado, considere que las entrada de referencia y los pesos *Vj* y *Wji* tienen un formato de 2:16, entonces el resultado debe ser presentado en un formato de 4:32, es decir, que el punto decimal se encuentra 32 lugares a la izquierda del bit menos significativo del resultado, por lo tanto, para ajustar el resultado al formato 2:16 hay que tomar los bits del 33 al 16.

4.2.- Simulación de la frecuencia de muestreo.

Presentaremos a continuación los resultados obtenidos por cada bloque lógico descrito en los pasos del algoritmo de aprendizaje de Retropropagación, empezamos con la simulación de la frecuencia de muestreo, el bloque lógico de ésta se muestra en la figura 4.1.

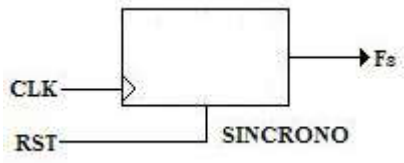


Figura 4.1.- Bloque lógico de frecuencia de muestreo

El bloque lógico de la figura es un contador como divisor de frecuencia, la frecuencia base del *FPGA* es de 50 Mhz. La simulación de este bloque de la frecuencia de muestreo es mostrada en la figura 4.2.

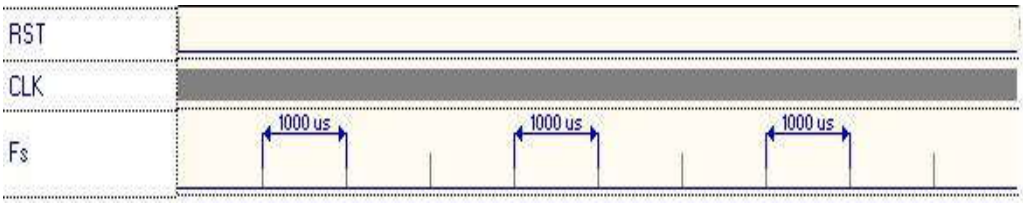


Figura 4.2.- Simulación de frecuencia de muestreo.

4.3.- Resultados y simulación del algoritmo de aprendizaje en VHDL.

El primer paso consiste en determinar o proponer valores pequeños de los pesos *Wji* y *Vj*, para este proceso de simulación utilizaremos un valor de 0.1 para ambos pesos.

El segundo paso del algoritmo de aprendizaje de la red neuronal autoajustable, en la figura 4.3 se muestra la estructura interna del bloque lógico de entrada. La descripción en software de síntesis VHDL, para la señal entrada de 18 bits y de acuerdo a la figura, la entrada se desfasa en x, x(t-1) y x(t-2), es decir, hay que tomar una muestra de los datos cada milisegundo, para esto utilizaremos registros como se muestra en la figura 4.3, donde la toma de los tres datos de muestra será con una frecuencia de 1 milisegundo.

Figura 4.3.- Bloque lógico de entrada.

La descripción en VHDL se muestra en el apéndice A, la simulación de este bloque lógico con un número representado en punto fijo con un formato de 2:16 se muestra en la figura 4.4.

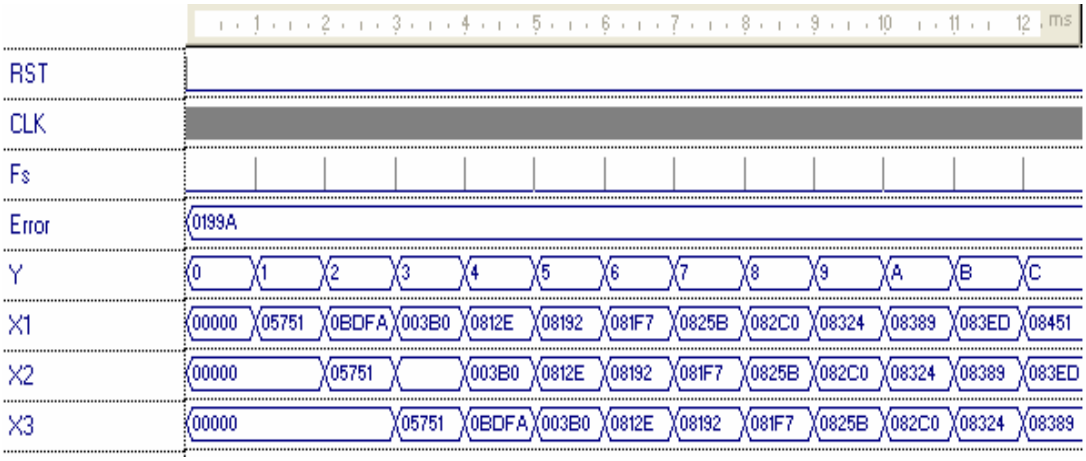


Figura 4.4.- Simulación del corrimiento de la señal de entrada.

En la figura 4.4 se observa que la señal de entrada es desfasada en dos tiempos anteriores. Para el calculo de *Sj* que es la entrada a las neuronas de la capa escondida de la red neuronal y es tercer paso del algoritmo de aprendizaje de retropropagación hacia atrás del red neuronal, que es la operación de la multiplicación de la señal de entrada *Xi* con el peso propuesto *Wji*, en la figura 4.5 se muestra el bloque lógico de la multiplicación de los parámetros de la entrada a la neurona de la capa escondida.

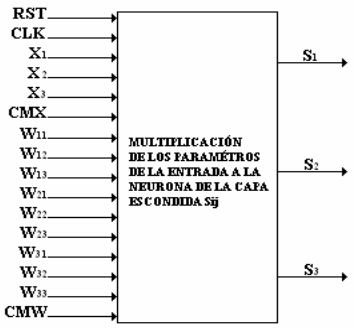


Figura 4.5.- Bloque lógico de la multiplicación de los parámetros de la entrada a la neurona de la capa escondida.

Por lo que respecta a los valores de entrada *X* y el peso *W* deben estar en un formato de 2:16, por lo que se necesita un formato de representación 2:16. En la tabla 1 contiene los valores de la señal *X* y *W* ajustados al formato 2:16.

Señal de Entrada	Números decimales positivos	Forma Binaria	Forma hexadecimal de números positivos
X_1	0.0144036	0000001110110000	03B0
X_2	0.7420980	1011110111111010	BDFA
X_3	0.3410840	0101011101010001	5751
W_{11}	0.0994660	0001100101110001	1971
W_{12}	0.0983280	0001100100101100	192C
W_{13}	0.0990260	0001100101011001	1959
W_{21}	0.0994640	0001100101110110	1976
W_{22}	0.0983240	0001100100101011	192B
W_{23}	0.0990330	0001100101011010	195A
W_{31}	0.1036200	0001101010000111	1A87
W_{32}	0.1015800	0001101000000001	1A01
W_{33}	0.1029900	0001101001011110	1A5E

Tabla 1.- Números decimales en formato 2:16

Para la simulación de las multiplicaciones utilizaremos los datos de la tabla 1, primeramente mostraremos las multiplicaciones con los resultados obtenidos:

$$S_1 = \frac{\overset{03B0}{x\ 1971}}{005DD0B0} + \frac{\overset{03B0}{x\ 192C}}{005CD240} + \frac{\overset{03B0}{x\ 1959}}{005D7830} = 001181B20$$

$$S_2 = \frac{\overset{BDFA}{x\ 1976}}{12E4FB3C} + \frac{\overset{BDFA}{x\ 195A}}{12D033E4} + \frac{\overset{BDFA}{x\ 192B}}{12AD52FE} = 03862821E$$

$$S_3 = \frac{\overset{5751}{x\ 1A87}}{90C45B7} + \frac{\overset{5751}{x\ 1A01}}{8DE9151} + \frac{\overset{5751}{x\ 1A5E}}{8FE49BE} = 01AE920C6$$

Los resultados de las multiplicaciones anteriores están representados en un formato de 4:32 bits, por lo que tenemos que ajustar el resultado a un formato de 2:16, entonces el resultado correcto esta en los bits del 33 al 16, por lo tanto el resultado correcto es el siguiente:

$$S_1 = 00118$$
$$S_2 = 03862$$
$$S_3 = 01AE9$$

Estos resultados se muestran en la simulación realizada del multiplicador acumulador para números positivos en el software de síntesis VHDL en la figura 4.6.

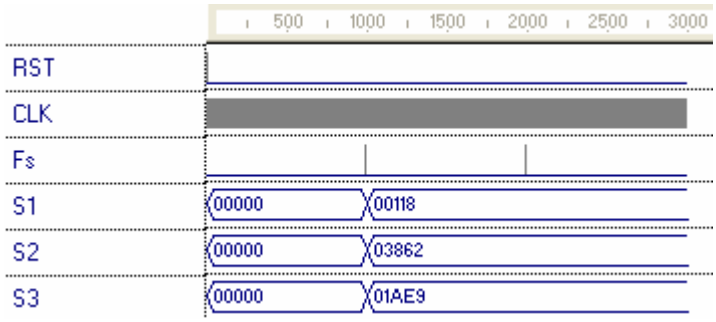


Figura 4.6.- Simulación del bloque lógico de la multiplicación de los parámetros de la entrada a la neurona de la capa escondida.

En la figura 4.6 se observa que los resultados de las multiplicaciones son los correctos. Pero los resultados mostrados son para puros números positivos, ahora mostraremos una simulación de multiplicación de números negativos, multiplicación de un número positivo y un número negativo, para estas operaciones es utilizado la multiplicación en el sistema de complemento 2, este sistema es utilizado por las computadoras, la multiplicación se efectúa de la misma forma que en la parte de números decimales, siempre y cuando el multiplicando y el multiplicador se pongan en forma binaria verdadera. Si los dos números a multiplicar son positivos, ya están en forma binaria verdadera y se multiplican de la forma tradicional. El producto resultante es positivo y se le coloca un bit de signo de 0. Cuando los números son negativos, deben estar representados en forma de complemento a 2, es decir, convertirlos a su forma

positiva y luego se multiplican. El producto es positivo por la regla de los signos y se le coloca un bit de signo de 0. Cuando uno de los números es positivo y el otro negativo, al negativo se le convierte primero en magnitud positiva tomando su complemento a 2. El producto estará en forma de magnitud verdadera. Sin embargo, el producto tiene que ser negativo, porque de acuerdo a la ley de los signos, los números tienen signos contrarios, por lo tanto, el producto se cambia a su forma de complemento a 2 y se le coloca un signo de 1. Ahora mostramos los mismos números decimales de la tabla 1 en su forma negativa en la tabla 2.

Señal de Entrada	Números decimales Negativos	Forma Binaria	Forma hexadecimal de números negativos
X_1	-0.0144036	1111111110001010000	3FC50
X_2	-0.7420980	1101000010000000110	34206
X_3	-0.3410840	1110101000101011111	3A8AF
W_{11}	-0.0994660	1111100110100011111	3E68F
W_{12}	-0.0983280	1111100110110101000	3E6D4
W_{13}	-0.0990260	1111100110101001111	3E6A7
W_{21}	-0.0994640	1111100110100010100	3E68A
W_{22}	-0.0983240	1111100110110101010	3E6D5
W_{23}	-0.0990330	1111100110101001100	3E6A6
W_{31}	-0.1036200	1111100101011110001	3E579
W_{32}	-0.1015800	1111100101111111111	3E5FF
W_{33}	-0.1029900	1111100101101000100	3E5A2

Tabla 2.- Números decimales Negativos en formato 2:16

Para demostrar la simulación de las multiplicaciones utilizaremos los datos de la tabla 2, primeramente realizaremos las multiplicaciones con los resultados obtenidos, en forma matemática sin utilizar el bloque lógico correspondiente:

$$S_1 = (3FC50)(3E68F) + (3FC50)(0192C) + (3FC50)(01959)$$

Para realizar la multiplicación de primer miembro que son números negativos de la ecuación anterior es necesario expresar estos en su forma de valor absoluto, es decir, aplicando el sistema de complemento a 2 como sigue:

Primer número negativo:

11 1111 1100 0101 0000	3FC50
00 0000 0011 1010 1111	complemento a 1.
_____1	complemento a 2.
00 0000 0011 1011 0000	03B0 número positivo

Segundo número negativo:

11 1110 0110 1000 1111	3E68F
00 0001 1001 0111 0000	complemento a 1.
_____1	complemento a 2.
00 0001 1001 0111 0001	1971 número positivo

$$\begin{array}{r} 03B0 \\ x\ \ 1971 \\ \hline 005DD0B0 \end{array}$$

$$S_{11} = 005DD0B0$$

En este caso el resultado es positivo, porque los números son negativos los dos.

Para el segundo término de la multiplicación tenemos un número positivo y un número negativo, en el caso del número negativo hay que cambiarlo al valor absoluto con el complemento a 2 como sigue:

Número negativo:

11 1111 1100 0101 0000	3FC50
00 0000 0011 1010 1111	complemento a 1.
_____1	complemento a 2.
00 0000 0011 1011 0000	03B0 número positivo

Ahora se realiza la multiplicación normalmente como sigue

$$S_{12} = \frac{\begin{array}{r} 03B0 \\ x\ \ 192C \end{array}}{005CD240}$$

Como los números tienen signos contrarios el producto es negativo por lo tanto, el producto se tiene que complementar a 2 como se muestra a continuación.

Producto:

00 0000 0000 0101 1100 1110 0010 0100 0000	005CD240
11 1111 1111 1010 0011 0001 1101 1011 1111	complemento a 1.
_____1	complemento a 2.
11 1111 1111 1010 0011 0001 1101 1111 0000	3FFA31DF0

El resultado de la multiplicación es el siguiente:

$$S_{12} = \frac{\begin{array}{r} 3FC50 \\ x\ \ 0192C \end{array}}{3FFA31EF0}$$

El siguiente término de la ecuación es la misma situación que la anterior por lo que se obtiene:

Producto:

00 0000 0000 0101 1101 0111 1000 0011 0000	005D7830
11 1111 1111 1010 0010 1000 0111 1100 1111	complemento a 1.
_____1	complemento a 2.
11 1111 1111 1010 0010 1000 0111 1101 0000	3FFA287D0

$$S_{13} = \frac{\begin{array}{r} 3FC50 \\ x\ \ 0192C \end{array}}{3FFA287D0}$$

$$S_1 = \frac{\begin{array}{r} 3FC50 \\ x\ \ 3E68F \end{array}}{005DD0B0} + \frac{\begin{array}{r} 3FC50 \\ x\ \ 0192C \end{array}}{3FFA31EF0} + \frac{\begin{array}{r} 3FC50 \\ x\ \ 01959 \end{array}}{3FFA287D0} = 3FFA37770$$

$$S_2 = \frac{x\ 1976^{BDFA}}{12E4FB3C} + \frac{x\ 195A^{BDFA}}{12D033E4} + \frac{x\ 192B^{BDFA}}{12AD52FE} = 03862821E$$

$$S_3 = \frac{x\ 1A87^{5751}}{90C45B7} + \frac{x\ 1A01^{5751}}{8DE9151} + \frac{x\ 1A5E^{5751}}{8FE49BE} = 01AE920C6$$

Los resultados de las multiplicaciones anteriores están representados en un formato de 4:32 bits, por lo que tenemos que ajustar el resultado a un formato de 2:16, entonces el resultado correcto esta en los bits del 33 al 16, por lo tanto el resultado correcto es el siguiente:

$$S_1 = 3FFA3$$
$$S_2 = 03862$$
$$S_3 = 01AE9$$

Estos resultados se muestran en la simulación realizada del multiplicador acumulador para números positivos en el software de síntesis VHDL en la figura 4.7.

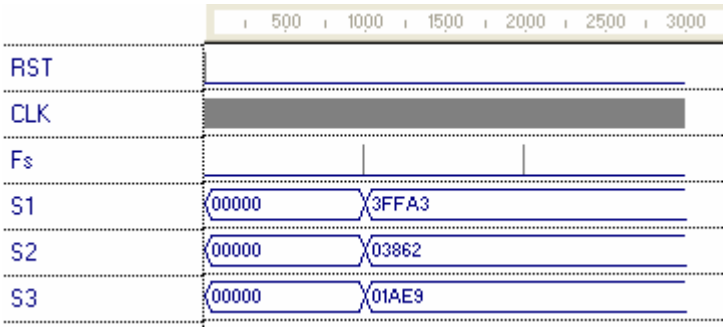


Figura 4.7.- Simulación del bloque lógico de la multiplicación de los parámetros de la entrada a la neurona de la capa escondida con números negativos.

El paso cuatro del algoritmo de aprendizaje de la red neuronal es la salida de las neuronas de la capa escondida y esta es conocida como la señal de activación. Esta señal de activación será aproximada por polinomios, por lo tanto el bloque lógico

se llamara bloque lógico de activación, y en la figura 4.8 se muestra la estructura de este bloque.

Figura 4.8.- Bloque lógico de la señal de activación

Para este bloque se realizo la aproximación por polinomios de la ecuación de la función de activación en Matlab con la siguiente ecuación de aproximación.

$$h = \sum_{k}^{n} a_n S_k \quad , \; a_n \neq 0$$

Desarrollando la ecuación obtenemos:

$$h = \sum_{k}^{n} a_n x^k = a_n S_1 + a_{n-1} S_1 + \cdots + a_2 S_1 + a_1 S_1 + a_0$$

La aproximación obtenida en la simulación fue con un polinomio de cuarto orden, por lo tanto podemos escribirlo de la siguiente forma:

$$h = \left[(0 + a4)S_1 + a3\right]S_1 + a2\}S_1 + a1\}S_1 + a1$$

Con el desarrollo anterior de ecuación de aproximación por polinomios de la función de activación, se obtienen los valores de los coeficientes del polinomio en Matlab y descriptos en VHDL por medio de una memoria ROM.

Con los valores obtenidos de S_1, S_2, y S_3 procedemos a comprobar los resultados, con los valores de los coeficientes o índices de la Memoria ROM con un formato de 2:16 de la variable *a* que se muestran en la tabla 3.

Coeficientes	Números decimales	Binaria	Hexadecimal
a_0	0.51678467	001000010001001100	0844C
a_1	0.37411499	000101111111000110	05FC6
a_2	-0.02581787	111111100101100100	3F964
a_3	-0.03166199	111111011111100101	3F7E5
a_4	0.00558472	000000000101101110	0016E

Tabla 3- Números decimales en formato 2:16

Procedemos al desarrollo de las operaciones como sigue:

Para $S_1 = 118$.

$$\begin{aligned} h_1 &= \left[(0+16E)118+3F7E5\right]118+3F964\}118+5FC6\}118+844C] = \\ &= \left[3F7E4\right]118+3F964\}118+5FC6\}118+844C] = \\ &= [3FFF8+3F964\}118+5FC6\}118+844C] = \\ &= [3FFF9+5FC6\}118+844C] = \\ &= [68+844C] = 084B4 \end{aligned}$$

En las multiplicaciones que se realizan en el desarrollo de las operaciones, como son los números de 18 bits, el resultado de las multiplicaciones es de números de 36 bits, por lo que tenemos que ajustar el resultado tomando los bits correspondientes del 33 al 16 para obtener los resultados correctos, que son los que toman en cada multiplicación.

Para S_2 y S_3 es el mismo procedimiento al anterior, los resultados por lo tanto son:

$$h_2 = 098FC$$

$$h_3 = 08E48$$

La simulación de la aproximación de polinomios de la función de activación se muestra en la figura 4.9.

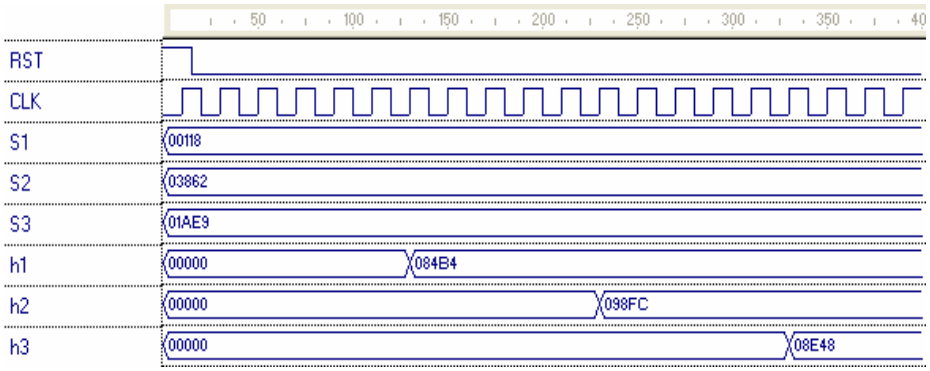


Figura 4.9.- Simulación del bloque de la señal de activación.

El paso 5 es la realización de la multiplicación de la salida de la neurona de la capa escondida por los pesos Vi. El bloque lógico para este paso del algoritmo de aprendizaje lo denominaremos como bloque lógico de entrada a la neurona de salida, la estructura interna del bloque lógico de entrada a la neurona de salida se muestra en la figura 4.10.

Figura 4.10.- Bloque lógico de entrada a la neurona de salida.

Por lo que respecta a los valores de la señal de activación hj y el peso V debe estar en un formato de 2:16, por lo que se necesita un formato de representación 2:16. En la tabla 2 contiene los valores de la señal de activación hj y V ajustados al formato 2:16.

Señal de Entrada	Números decimales positivos	Binario 2:16	Representación Hexadecimal de números positivos
h_1	0.19638	0011001001000110	84B4
h_2	0.19568	0011001000011000	3218
h_3	0.50000	0100000000000000	4000
V_1	0.20364	0011010000100001	3421
V_2	0.20351	0011010000011001	3419
V_3	0.27047	0100010100111110	453E

Tabla 4.- Números decimales positivos en formato 2:16

Para la simulación de las multiplicaciones utilizaremos los datos de la tabla 4, primeramente mostraremos las multiplicaciones con los resultados obtenidos:

$$r = \frac{x\ \ 3246}{A3CB306} + \frac{x\ \ 3218}{A31C458} + \frac{x\ \ 4000}{114F8000} = 025BDF75E$$

El resultado de la multiplicación anterior está representado en un formato de 4:32 bits, por lo que tenemos que ajustar el resultado a un formato de 2:16, entonces el resultado correcto esta en los bits del 33 al 16, por lo tanto el resultado correcto es el siguiente:

$$r = 025BD$$

Este resultado se muestra en la simulación realizada en el software de síntesis VHDL del multiplicador acumulador de la señal de salida de la neurona de la capa de salida en la figura 4.11.

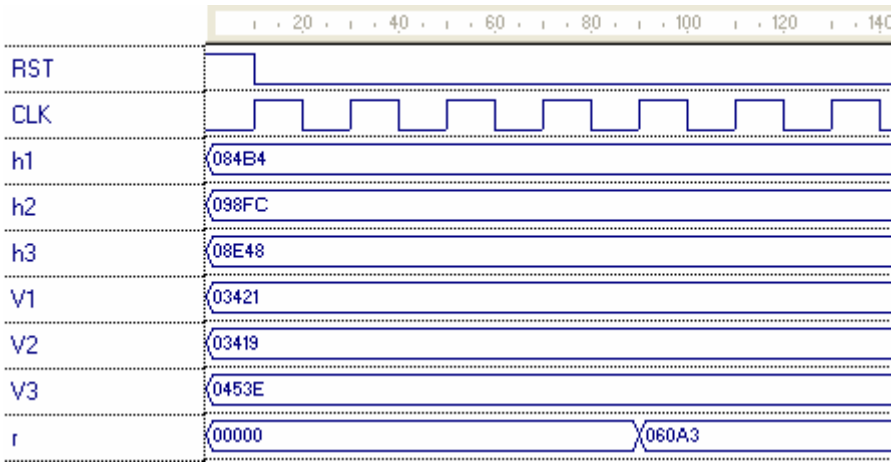


Figura 4.11- Simulación de la MAC de la señal de salida de la neurona de la capa de salida.

En esta parte del algoritmo de aprendizaje realizaremos ahora multiplicaciones de números negativos, para verificar si la descripción en VHDL de este bloque es el correcto, utilizaremos los valores negativos de los números positivos de la tabla 4, en la tabla 5 mostraremos sus valores negativos.

Señal de Entrada	Números decimales positivos	Binario 2:16	Representación Hexadecimal de números negativos
h_1	-0.19638	1111001101010001l0	3CDBA
h_2	-0.19568	1111001101111101000	3CDE8
h_3	-0.50000	1111000000000000000	3C000
V_1	-0.20364	1111001011110111111	3CBDF
V_2	-0.20351	1111001011111100111	3CBE7
V_3	-0.27047	1110111010110000l0	3BAC2

Tabla 5.- Números decimales negativos en formato 2:16

Para realizar una simulación de multiplicaciones con números positivos y negativos utilizaremos los datos de la tabla 3 y de la tabla 4, primero multiplicaremos dos números negativos y posteriormente un número negativo con un número positivo como se muestra a continuación:

$$r = \frac{3CDBA \;\; x \;\; 3CBDF}{A3CB306} + \frac{3419 \;\; x \;\; 3CDE8}{FF5CE3BA8} + \frac{3BAC2 \;\; x \;\; 4000}{3EEB08000}$$

Para realizar la multiplicación de primer miembro que son números negativos de la ecuación anterior es necesario expresar estos en su forma de valor absoluto, es decir, aplicando el sistema de complemento a 2 como sigue:
Primer número negativo:

11 1100 1101 1011 1010	3CDBA	
00 0011 0010 0100 0101	complemento a 1.	
_____1	complemento a 2.	
00 0011 0010 0100 0110	03246	número positivo

Segundo número negativo:

11 1100 1011 1101 1111	3CBDF	
00 0011 0100 0010 0000	complemento a 1.	
_____1	complemento a 2.	
00 0011 0100 0010 0001	3421	número positivo

Como los números son negativos el producto por lo tanto es positivo, la multiplicación se realiza de manera normal como sigue:

$$r_1 = \frac{3246 \;\; x \;\; 3421}{00A3CB306}$$

Para el tercer término de la multiplicación tenemos un número positivo y un número negativo, en el caso del número negativo hay que cambiarlo al valor absoluto con el complemento a 2 como sigue:

Número negativo:

11 1100 1101 1110 1000	3CDE8
00 0011 0010 0001 0111	complemento a 1.
_____1	complemento a 2.
00 0011 0010 0001 1000	03218 número positivo

Ahora se realiza la multiplicación normalmente como sigue

$$r_2 = \frac{\begin{array}{r} 3419 \\ x\ \ 3218 \end{array}}{00A31C458}$$

Como los números tienen signos contrarios el producto es negativo por lo tanto, el producto se tiene que complementar a 2 como se muestra a continuación.

Producto:

0000 0000 1010 0011 0001 1100 0100 0101 1000	A31C4580
1111 1111 0101 1100 1110 0011 1011 1010 0111	complemento a 1.
_____1	complemento a 2.
1111 1111 0101 1100 1110 0011 1011 1010 1111	FF5CE3BA8

El resultado de la multiplicación de un número positivo y un número negativo es el siguiente:

$$r_2 = \frac{\begin{array}{r} 3419 \\ x\ \ 3CDE8 \end{array}}{FF5CE3BA8}$$

Para el tercer término de la multiplicación tenemos un número negativo y un número positivo, en el caso del número negativo hay que cambiarlo al valor absoluto con el complemento a 2 como sigue:

Número negativo:

11 1011 1010 1100 0010	3BAC2
00 0100 0101 0011 1101	complemento a 1.
_____1	complemento a 2.
00 0100 0101 0011 1110	0453E número positivo

Ahora se realiza la multiplicación normalmente como sigue

$$r_3 = \frac{\begin{array}{r} 453E \\ x\ \ 4000 \end{array}}{114F8000}$$

Como los números tienen signos contrarios el producto es negativo por lo tanto, el producto se tiene que complementar a 2 como se muestra a continuación.

Producto:

0000 0001 0001 0100 1111 1000 0000 0000 0000	114F8000
1111 1110 1110 1011 0000 0111 1111 1111 1111	complemento a 1.
_____1	complemento a 2.
1111 1110 1110 1011 0000 1000 0000 0000 0000	FEEB08000

El resultado de la multiplicación de un número positivo y un número negativo es el siguiente:

$$r_3 = \frac{\begin{array}{r} 3BAC2 \\ x\ \ 4000 \end{array}}{FEEB08000}$$

El producto final de la multiplicación de la función de activación y el peso Vij se muestra a continuación:

$$r = \frac{\begin{array}{r} 3CDBA \\ x\ \ 3CBDF \end{array}}{A3CB306} + \frac{\begin{array}{r} 3419 \\ x\ \ 3CDE8 \end{array}}{FF5CE3BA8} + \frac{\begin{array}{r} 3BAC2 \\ x\ \ 4000 \end{array}}{FEEB08000} = 0FEEBB6EAE$$

El resultado de la multiplicación anterior está representado en un formato de 4:32 bits, por lo que tenemos que ajustar el resultado a un formato de 2:16, entonces el resultado correcto esta en los bits del 33 al 16, por lo tanto el resultado correcto es el siguiente:

$$r = 3EEBB$$

Este resultado es mostrado en la simulación realizada en el software de síntesis VHDL del multiplicador acumulador de la señal de salida de la neurona de la capa de salida en la figura 4.12.

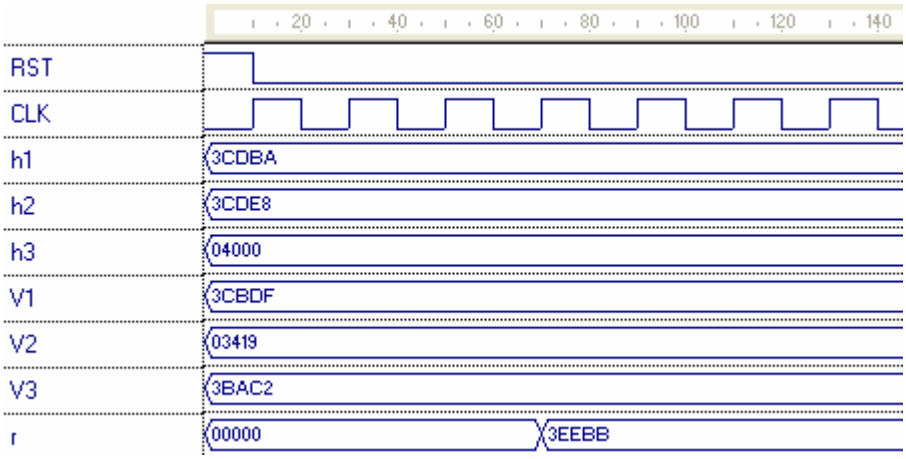


Figura 4.12.- Simulación de la MAC de la señal de salida de la neurona de la capa de salida con números negativos.

Para el siguiente paso, se ocupa otra vez la aproximación de polinomios de la señal de activación, como se muestra en el bloque de la figura 4.13, para el cálculo de u.

Figura 4.13.- Bloque Lógico para el cálculo de u.

$$u = \left[(0 + a4)r + a3\right]r + a2\}r + a1\}r + a0$$

Desarrollamos la ecuación de aproximación por polinomios utilizando el la regla de Horner obtenemos:

Para r = 60A3

$$\begin{aligned} u &= \left[(0 + 16E)60A3 + 3F7E5\right]60A3 + 3F964\}60A3 + 5FC6\}60A3 + 844C] = \\ &= \left[3F75B\right]60A3 + 3F964\}60A3 + 5FC6\}60A3 + 844C] = \\ &= [3FCBD + 3F964\}60A3 + 5FC6\}60A3 + 844C] = \\ &= [3FC47 + 5FC6\}60A3 + 844C] = \\ &= [22BF + 844C] = 0A719 \end{aligned}$$

La simulación en VHDL de este paso se muestra en la figura 4.14

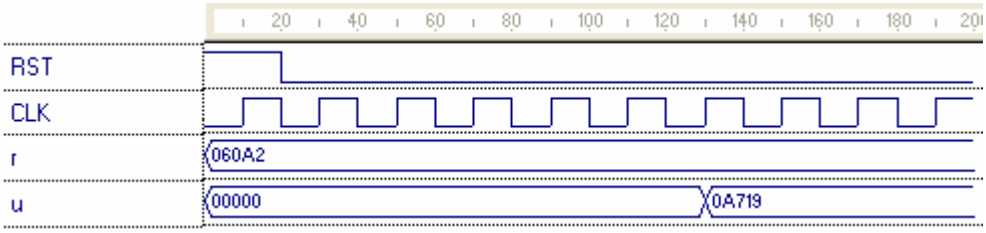


Figura 4.14.- Simulación de la operación para el calculo de u.

El paso 6 del algoritmo de aprendizaje de retropropagación o propagación hacia atrás, es el cálculo de la actualización de los pesos *Vj*. El bloque general que se utilizo para este proceso se muestra en la figura 4.15.

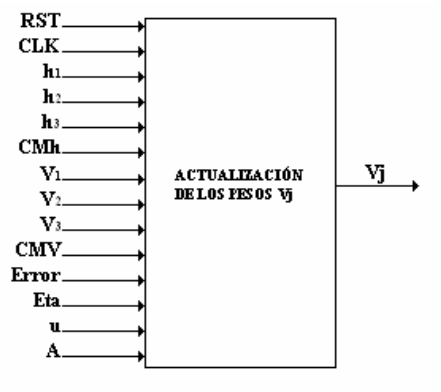


Figura 4.15.- Bloque de actualización de pesos Vj.

El bloque de la figura 4.15 representa en forma digital la ecuación:

$$v_j(t+1) = v_j(t) + \eta\delta_1 h_j$$

Donde:

$$\delta_1 = (e_y)\left[u(1-u)\right]$$

Desarrollando la ecuación de la actualización de los pesos *Vj* obtenemos:

$$v_1 = v_1 + \eta e_y u(A-u)h_1$$
$$v_2 = v_2 + \eta e_y u(A-u)h_2$$
$$v_3 = v_3 + \eta e_y u(A-u)h_3$$

Los valores utilizados para las operaciones de las ecuaciones anteriores, así como para la simulación en VHDL para este Bloque Lógico se muestran Tabla 6 con un formato de 2:16.

Entradas	Decimal	Binario	Hexadecimal
V1	0.203628	0000111010000100001	3421
V2	0.203506	0000111010000011001	3419
V3	0.270477	0001000101001111110	453E
Eta	1	0100000000000000000	10000
Error	0.1	0000011001100110 10	199A
A	1	0100000000000000000	10000
u	0.652725	0010100111000110 01	A719
h_1	0.518371	0010000100101101 00	84B4
h_2	0.597595	0010011000111111 00	98FC
h_3	0.555786	0010000111001001 000	8E48

Tabla 6.- Valores utilizados para la simulación.

A continuación utilizaremos las ecuaciones para el cálculo de la actualización de los pesos *Vj* como sigue:

$$\delta_1 = e_y u(1-u) =$$
$$\delta_1 = (199A)\left[A719\,(10000 - A719)\right] =$$
$$\delta_1 = (199A)\left[3A07\right] = 005CD$$
$$v_1 = v_1 + Eta\,\delta_1 h_1 =$$
$$v_1 = 3421 + (10000)(005CD)(84B4) =$$
$$v_1 = 3421 + 00301 = 03722$$
$$v_2 = v_2 + Eta\,\delta_1 h_2 =$$
$$v_2 = 3419 + (10000)(005CD)(098FC) =$$
$$v_2 = 3419 + 00377 = 03790$$
$$v_3 = v_3 + Eta\,\delta_1 h_3 =$$
$$v_3 = 3421 + (10000)(005CD)(8E48) =$$
$$v_3 = 453E + 00339 = 04877$$

Las multiplicaciones realizadas en las ecuaciones anteriores dan un resultado de 36 bits, es decir, en un formato de 4:32, por lo tanto tenemos que ajustar el

resultado a un formato de 2:16, de 18 bits, para este el resultado de cada multiplicación se tomaran los bits del 33 al 16.

La simulación en VHDL en forma digital de las ecuaciones de la actualización de los pesos *Vj*, se muestra en la figura 4.16.

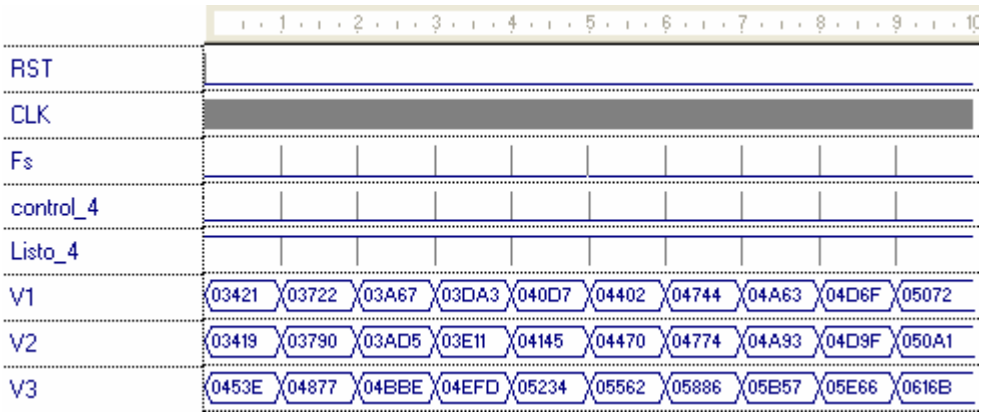


Figura 4.16.- Simulación de actualización de pesos Vj.

En la figura 4.16 se observa que la actualización de los pesos Vj, se van actualizando cada milisegundo. Entonces para obtener los siguientes resultados que se muestran en la simulación de la figura 4.16, es necesario realizar todas las operaciones que se realizaron anteriormente para verificar el resultado.

El paso 7 del algoritmo de aprendizaje de retroprogación o propagación hacia atrás, es el cálculo de la actualización de los pesos *Wj*i. El bloque general que se utilizo para este proceso se muestra en la figura 4.17.

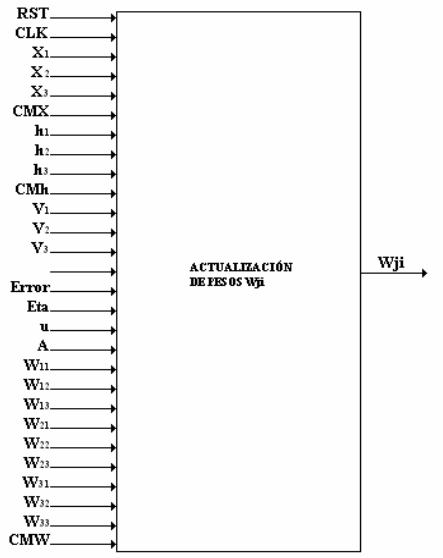


Figura 4.17.- Bloque de actualización de pesos Wji.

El bloque Lógico de la figura 4.17 representa en forma digital la ecuación:

$$w_{ji}(t+1) = w_{ji}(t) + \eta\delta_2 x_i$$

Donde: $\delta_1 = (e_y)\left[u\,(1 - u)\right]$

$$\delta_2 = \delta_1 v_j h_j (1 - h_j)$$

Los valores utilizados para las operaciones de las ecuaciones anteriores, así como para la simulación en VHDL para este Bloque Lógico se muestran Tabla 7 con un formato de 2:16.

Entradas	Decimal	Binario (2:16)	Hexadecimal
W_{11}	0.099472	0000011000101110111	1977
W_{12}	0.098327	0000011001001011 00	192C
W_{13}	0.099014	0000011001010110 01	1959
W_{21}	0.099456	0000011001011101 10	1976
W_{22}	0.098312	0000011001001010 11	192B
W_{23}	0.099029	0000011001010110 10	195A
W_{31}	0.103622	0000011010100001 11	1A87
W_{32}	0.101577	0000011010000000 01	1A01
W_{33}	0.102996	0000011010010111 10	1A5E
V1	0.203628	0000110100001000 01	3421
V2	0.203506	0000110100000110 01	3419
V3	0.270477	0001000101001111 10	453E
Eta	1	0100000000000000 00	10000
Error	0.1	0000011001100110 10	199A
1	1	0100000000000000 00	10000
u	0.652725	0010100111000110 01	A719
h_1	0.518371	0010000100101101 00	84B4
h_2	0.597595	0010011000111111 00	98FC
h_3	0.555786	0010000111001001 00	8E48
X_1	0.0144036	0000001110110000	03B0
X_2	0.7420980	1011110111111010	BDFA
X_3	0.3410840	0101011101010001	5751

Tabla 7.- Valores utilizados para la simulación de este bloque lógico.

A continuación se muestran todas las operaciones realizadas para el cálculo de la actualización de los pesos Wji de la ecuación (4.6)

.

$$\delta_1 = e_y u(1-u) =$$
$$\delta_1 = (199A)\left[A719(10000 - A719)\right] =$$
$$\delta_1 = (199A)\left[3A07\right] = 005CD$$
$$W_{11} = 1977 + (10000)(005CD)(3421)(84B4)(10000 - 84B4)(003B0) =$$
$$W_{11} = 1977 + (12E)(84B4)(7B4C)(003B0) =$$
$$W_{11} = 1977 + (9C)(7B4C)(003B0) =$$
$$W_{11} = 1977 + (4B)(003B0) = 1977 + 00001 = 1978$$

$$W_{12} = 192C + (10000)(005CD)(3421)(84B4)(10000 - 84B4)(003B0) =$$
$$W_{12} = 192C + (12E)(84B4)(7B4C)(003B0) =$$
$$W_{12} = 192C + (9C)(7B4C)(003B0) =$$
$$W_{12} = 192C + (4B)(003B0) = 192C + 00001 = 192D$$

$$W_{13} = 1959 + (10000)(005CD)(3421)(84B4)(10000 - 84B4)(003B0) =$$
$$W_{13} = 1959 + (12E)(84B4)(7B4C)(003B0) =$$
$$W_{13} = 1959 + (9C)(7B4C)(003B0) =$$
$$W_{13} = 1959 + (4B)(003B0) = 1959 + 00001 = 195A$$

$$W_{21} = 1976 + (10000)(005CD)(3419)(98FC)(10000 - 98FC)(BDFA) =$$
$$W_{21} = 1976 + (12E)(98FC)(6704)(BDFC) =$$
$$W_{21} = 1976 + (B4)(6704)(BDFA) =$$
$$W_{21} = 1976 + (48)(BDFA) = 1976 + 0035 = 19AB$$

$$W_{22} = 192B + (10000)(005CD)(3419)(98FC)(10000 - 98FC)(BDFA) =$$
$$W_{22} = 192B + (12E)(98FC)(6704)(BDFC) =$$
$$W_{22} = 192B + (B4)(6704)(BDFA) =$$
$$W_{22} = 192B + (48)(BDFA) = 192B6 + 0035 = 1960$$

$$W_{23} = 195A + (10000)(005CD)(3419)(98FC)(10000 - 98FC)(BDFA) =$$
$$W_{23} = 195A + (12E)(98FC)(6704)(BDFC) =$$
$$W_{23} = 195A + (B4)(6704)(BDFA) =$$
$$W_{23} = 195A + (48)(BDFA) = 195A + 0035 = 198F$$

$$W_{31} = 1A87 + (10000)(005CD)(453E)(8E48)(10000 - 8E48)(5751) =$$
$$W_{31} = 1A87 + (191)(8E48)(71B8)(5751) =$$
$$W_{31} = 1A87 + (DE)(71B8)(5751) =$$
$$W_{31} = 1A87 + (62)(5751) = 1A87 + 0021 = 1AA8$$

$$W_{32} = 1A01 + (10000)(005CD)(453E)(8E48)(10000 - 8E48)(5751) =$$
$$W_{32} = 1A01 + (191)(8E48)(71B8)(5751) =$$
$$W_{32} = 1A01 + (DE)(71B8)(5751) =$$
$$W_{32} = 1A01 + (62)(5751) = 1A01 + 0021 = 1A22$$

$$W_{33} = 1A5E + (10000)(005CD)(453E)(8E48)(10000 - 8E48)(5751) =$$
$$W_{33} = 1A5E + (191)(8E48)(71B8)(5751) =$$
$$W_{33} = 1A5E + (DE)(71B8)(5751) =$$
$$W_{33} = 1A5E + (62)(5751) = 1A5E + 0021 = 1A7F$$

Las multiplicaciones anteriores son de 18 bits cada una, por lo tanto nos da como resultado una multiplicación de 36 bits, por lo que todas estas, están ajustadas al formato 2:16, por lo que el resultado que se tomo es del bit 33 al 16, de los 36 bits. La simulación de estas operaciones del bloque de actualización de pesos Wji en VHDL se muestra en la figura 4.18.

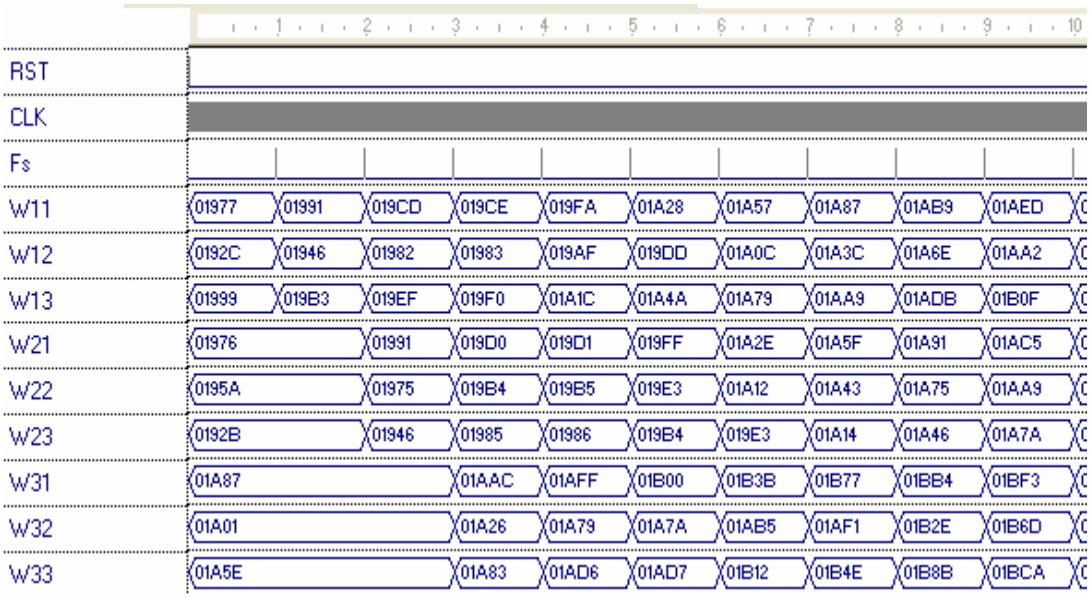


Figura 4.18.- Simulación de las operaciones de la actualización de pesos Wji.

Se observa de la figura 4.18 que el valor de los pesos *Wji*, son los mismos que se calcularon anteriormente, es decir que la descripción de la ecuación de la actualización de los pesos en VHDL es la correcta.

Ahora el paso final es la unión de todos los bloques lógicos que se construyeron para la descripción en VHDL del algoritmo de aprendizaje de Retropropagación de la red neuronal artificial Autoajustable, que es el bloque que se presenta en la figura 4.19.

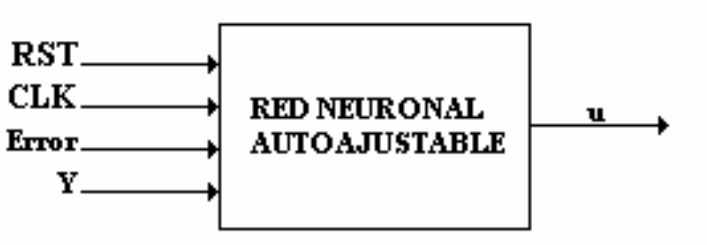


Figura 4.19.- Red Neuronal artificial Autoajustable.

En la figura 4.20 se muestra la simulación en VHDL del algoritmo de aprendizaje de Retropropagación que es utilizado para la Red Neuronal Artificial Autoajustable. Para esta simulación se utilizaron los valores para los pesos *Wij*, *Vj*, eta y el error, los utilizados en las simulaciones de los bloques lógicos anteriores, para los valores de X se utilizó una memoria ROM con 15 valores de 18 bits aleatorios, que son muestreados cada milisegundo. En la figura 4.20 se observa que la salida de la variable *u* se calcula cada milisegundo, dando valores diferentes, a su vez se puede observar como se van actualizando los pesos *Wji* y *Vj*.

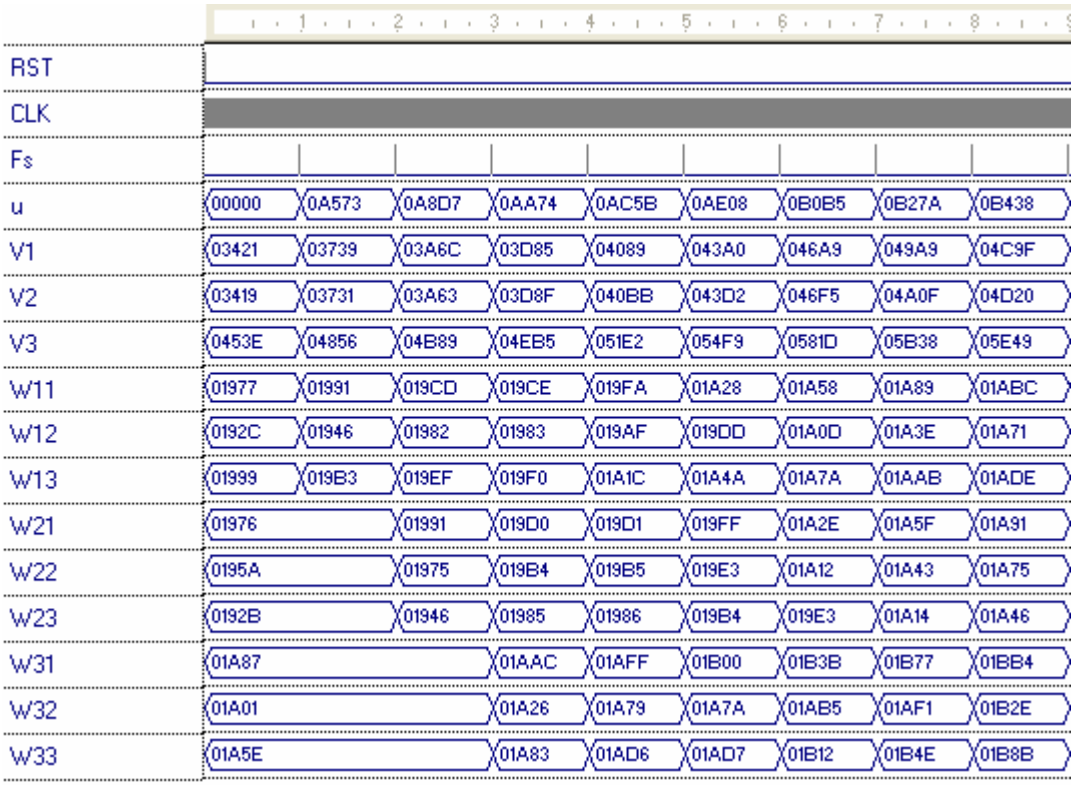


Figura 4.20.- Simulación de la Red Neuronal Artificial Autoajustable en VHDL.

4.4 Conclusiones.

La implementación en el FPGA del modelado del algoritmo de aprendizaje de la red neuronal artificial autoajustable, que hasta el momento se tiene, permite obtener un control inteligente dedicado para un servomotor de corriente directa.

Las redes neuronales son una estructura bien definida, y el algoritmo muy apropiado para modelar cualquier tipo de sistema. Por lo tanto una red neuronal es una herramienta para modelar una planta cualquiera que contenga no linealidades, permitiendo que los parámetros sean variados, también permite un rango grande no linealidades.

Las redes neuronales son un área de la inteligencia artificial que ha evolucionado en los últimos años, por lo que ha despertado mucho interés, en esta área. Se encontró que las redes neuronales son capaces de resolver problemas de control muy complejos, donde con las herramientas de diseño se pueden modelar e implementar en un FPGA.

Las redes neuronales artificiales son una opción muy poderosa para desarrollar controles inteligentes que tomen sus propias decisiones, esto ocurre por medio de su entrenamiento o algoritmo de aprendizaje.

REFERENCIAS.

S.-I. Amari. 1980. *Topographic organization of nerve fields*. Bull. Math. Biology, vol. 42, pp. 339-364.

Cui, X; Shin, K.G. 1992. *Direct Control and Coordination Using Neural Networks*, IEEE Transactions on Systems, Man and Cybernetics, Vol. 23,No 3, pp. 686-697.

D. L. Chester.1990. Why two hidden layers are better than one. *In Proceedings of the International Joint Conference on Neural Network*s, vol. 1, pp. 265-268. Erlbaum.

G. Cybenko. 1989. *Approximation by superpositions of a sigmoidal function*. Mathematics of Control, Signals, and Systems, Vol. 2, pp. 303-314.

Bounds, D., Lloyd, P., Mathew, B., Waddell, A. 1988. *Multilayer Perceptron for the Diagnosis of Low Back Pain*. *IEEE International Conference on Neural Network*, San Diego.

Gian Paolo Drago, Sandro Ridella. 1992. *Statistically Controlled Activation Weight Initialization (SCAWI)*. IEEE Transactions on Neural Networks, Vol. 3, NO. 4, pp. 627-631.

R. M. Gray. 1984. *Vector quantization*. IEEE ASSP Magazine, vol. 1, pp. 4- 29.

Jang, J.S.R. and Sun, C.T. 1995. *Neuro-Fuzzy Modeling nd Control.* Proc. IEEE, vol.83, No.3 pp. 378-406.

T. Kohonen. 1997. *Self-Organization and Associative Memory*. Springer-Verlag. Berlin.

T. Kohonen. 1988. *An Introduction to Neural Computing, Neural Network*, vol.1, no. 1, pp. 3-16 Pergamon Press.

Teuvo Kohonen. 1990. *The Self-Organizing* Map. Proceedings of the IEEE, Vol. 78, No. 9, pp. 1464-1480.

Y. Linde, A. Buzo, R. M. Gray. 1980. *An algorithm for vector quantization*. IEEE Trans. Communication, vol. COM-28 pp 84-95.

R. Lippmann. April 1987. *An introduction to computing with neural nets. IEEE Acoustics, Speech and Signal Processing Magazine*, Vol. 4, pp. 4-22.

M. Minsky, S. Papert. 1969. *Perceptrons*. Cambridge, MA: MIT Press.

K. S. Narendra and K. Parthasarathy. 1991. *Gradient methods for the optimization of dynamic systems containing neural networks*. IEEE Transactions on Neural Networks, pp. 252-262.

O. Nerrand, P. Roussel-Ragot, L. Personnaz, G. Dreyfus, S. Marcos. 1993. *Neural networks and nonlinear adaptive filtering: unifying concepts and new algorithms*. Neural Computation. vol. 5, pp. 165-199.

F. Rosenblatt. 1958. The perceptron: *A probabilistic model for information storage and organization in the brain*. Psychological Review. 65;386-408.

Rumelhart, D., McClelland, J. 1986. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vol. I, Chap. 8, MIT, Cambridge, Mass.

P. J. Werbos, Beyond. 1974. regression: *New tools for prediction and analysis in the behavioral sciences.* Masters thesis, Harvard University.

Widrow, B., Hoff., M.. 1960. *Adaptive Switching Circuits*, IRE WESCON Conv. Records, Part. 4, pp. 96 -104

Lodewyk F. A.Wessels, Etienne Barnard. 1992. *Avoiding False Local Minima by Proper Initialization of Connections*. IEEE Transactions on Neural Networks, Vol. 3, No. 6, pp. 899-905.

APÉNDICES

En los apéndices, el código mostrado muestra únicamente el código del tema explicado y/o utilizado en el capítulo correspondiente. Aquí se muestra nada más el código de tres bloques lógicos VHDL y en el CD anexo pueden encontrarse los siguientes bloques lógicos completos así como los módulos del algoritmo de aprendizaje en VHDL requeridos para reproducir los análisis presentados en el presente trabajo.

Apéndice A: Código VHDL del bloque lógico de la frecuencia de muestreo.

Apéndice B: Código VHDL del bloque lógico de Multiplicador acumulador de la entrada a la neurona de la capa escondida**.**

Apéndice C: Código VHDL del bloque lógico de entrada a la neurona de entrada.

.

"FLEXIBLE MANUFACTURING SYSTEM SIMULATION USING PETRI NETS"

Carlos G. Mireles Preciado, Alfonso Noriega Ponce and Gerardo Leyva Soto
Facultad de Ingeniería, Universidad Autónoma de Querétaro
Centro Universitario. 76010, Querétaro, Qro.
e-mail: carlos.mireles@effem.com
anoriega@uaq.mx

ABSTRACT

We are presenting Petri Nets (PN) as a graphic tool for modeling discrete dynamical systems and their application to the modeling of flexible manufacturing systems. The advantage of Petri nets for evaluating and analysis of the model is presented as well. Practical examples are presented and finally a detail explanation of the software developed for simulation.

1. INTRODUCTION

In this work we are dealing with flexible manufacturing systems. The approach of Petri Nets is proposed, which are suitable for discrete dynamical systems. Systems like these can be completed analyzed and then be useful for practical implementation. A Flexible Manufacturing System (FMS) is formed by work stations (machines, tools, CNN machines, assembly stations, etc) linked by an automated transport system which is operated by a computer. These kind of systems can also have a storage for both final product and tools. We present a short description of the FMSs, then the mathematical structure of a Petri Net. After this we present some applications to the modeling of FMSs. Finally we discussed about a simulation package and some application examples.

2. BASIC NOTIONS OF PETRI NETS

A Petri Net (PN) is an oriented graph with two types of nodes: Places and ransitions. These are represented by circles and bars respectively. Nodes are related with each other by directed arcs which connect them to the Transitions. Figure 1 shows a simple representation of a PN. Here we can see how Places and Transitions are connected to each other by the directed arcs.

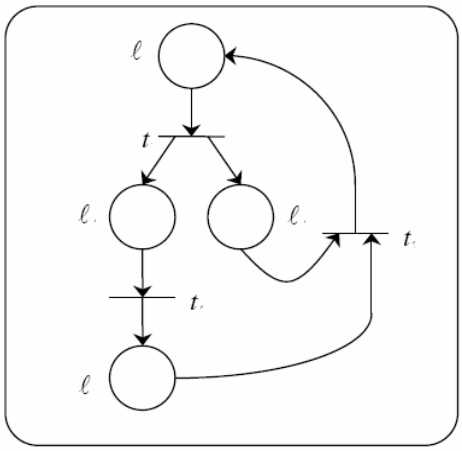


Figure 1. Petri Net

A transition has input & output places. On the PN of figure 1, transition t_1 has an input place (ℓ_1) and two output places (ℓ_2 and ℓ_3). In occasions an input place to a transition can be also an output place, then we have a loop.

A PN represents a dynamical behavior by marks. A place can have zero, one o several marks, represented by points within the circles. A PN with marks is called as Marked PN. The set of marked places represent its state in a certain given moment. The initial marking is defined as the set of places which

are marked and the beginning of an execution of the PN.

A transition is validated when all its input places are marked. The trigger of a transition makes the marking of the PN different. The trigger process consist in removing a mark to all of the input places and to add one mark to every output place. See figure 2.

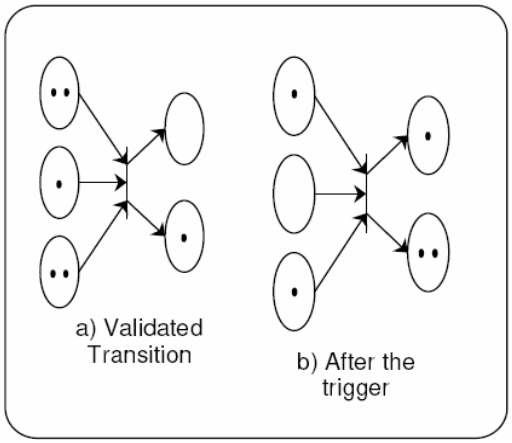


Figure 2

When modeling sequential systems with PN, places, transitions and marks take certain meaning accordingly with the system which is being modeled. Places are resources, operations, partial states; Marks are the availability of those resources, order of execution of certain operations. A transition can be the beginning of the ending of a new event. A PN which has an associated meaning for their places, transitions and marks is called an Interpreted PN.

3. MODELLING OF SYSTEMS

When automating, PN can be widely used in the design of sequential auto-mechanisms for production control systems. En el área de automatización las RP pueden ser ampliamente utilizadas en el diseño de automatismos secuenciales para el control de sistemas de producción. Presentamos aquí ejemplos que abordan esos dos niveles de automatización de procesos (3). (6).

3.1 Manufacturing Pieces

We need control a machinery process for metalic parts which enter to the machinery center by a transport system to which the parts are firmely attached. The transport system carries on when the variable P=1. The button M starts the system making the first part to enter into the machinery center. Some holes needs to be done on the parts as shown in figure 3 y they are done by using a couple of drilling machines, each of those able to use any of two drills, one with diameter d1 and the other with diameter d_2 ($d_1 < d_2$). To use a drill tool a logic 1 should be assigned to the correspondent variable B_1 or b_1, B_2 or b_2 (see figure). The logic value 1 on the before mentioned variables does not need to remain present. We consider for the analysis that the tool mounting on the drilling machine is quite fast. The drilling machines go forward or go backward when applying a logic value 1 to the correspondient variable I_1 or D_1, I_2 or D_2.

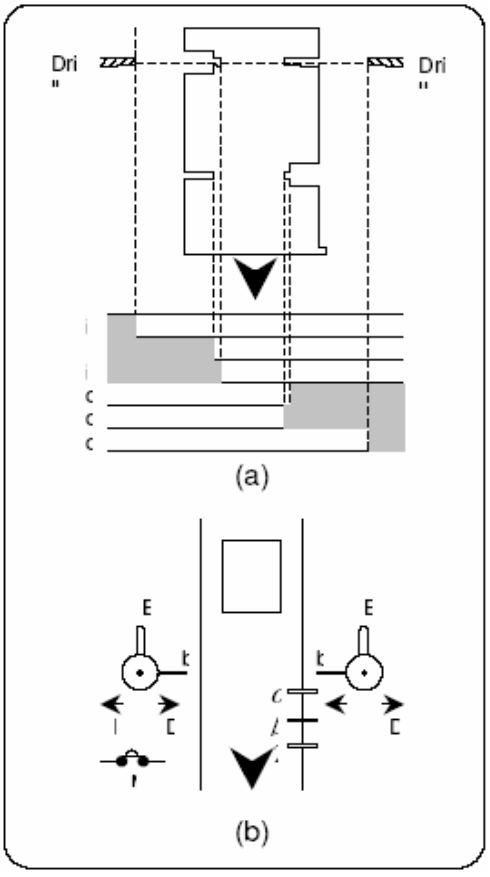


Figure 3. Machinery System

Point drill tools positions are indicated by a group of contacts associated to the displacement mechanism of each drilling machine. The correspondient signals (i1, i2, i3,d1, d2, d3) have a value of 1 in the shadow areas of figure 3. When a couple of holes needs to be done, the thiner will be done first.

To locate the position of the two holes we have two contacts (a & b) which are actuated by the part itself. The contact Y indicates to the system that a part is leaving the machinery center. If when Y is actuated m=1 then the machinery cycle is started when another part arrives to the machinery center (7).

Figure 4 sahows a proposal to work out the above situation.

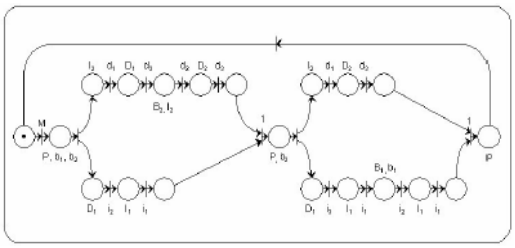


Figure 4. PN of the Machinery Center.

3.2 Flexible Manufacturing Cell

The purpose of this example is to show how many events can happen in a concurrent or asyncronous way. Consider a Flexible Manufacturing Cell (FMC), including three robots (R1, R2 y R3) and three transport conveyors (T1, T2 y T3) in a traignular configuration like the one shown in figure 5.

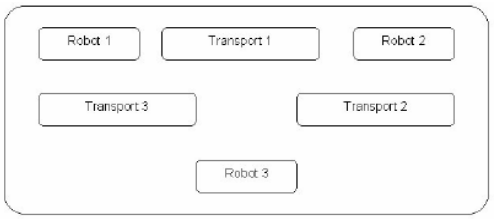


Figure 5. Flexible Manufacturing Cell

Any of the trasnport and two near robots are needed to make an assembly task. Each transport system request to the robot on the left, and after that request to the one on the right. Inmediately after the task is finished, the transpport system releases both of the robots.

Request or release a robot are two events which can happen at the same time or a different times (which is concurrent and asyncronous). As an example, the three trasnposrt systems can at the same time request to its correspondient left hand side robot (concurrency). Similarly, the relesase of two robots can happen when an assembly task is finished but the time of to complete the task can not be determined in advance with enough precision due to delays and errors (asyncronous). As a result, the problem of modeling this kind of behaviours is solve by using the simulation with PN.

The Places and Transitions for the above system are listed in the following table:

I_1	T_1	Request its left hand side robot R_1 (I_{10})
I_4	T_2	Request its left hand side robot R_2 (I_{11})
I_7	T_3	Request its left hand side robot R_3 (I_{12})
I_2	T_1	Request its right hand side robot R_2 (I_{11})
I_5	T_2	Request its right hand side robot R_3 (I_{12})
I_8	T_3	Request its right hand side robot R_1 (I_{10})
I_3	T_1	Both robots R_1 y R_2 are working
I_6	T_2	Both robots R_2 y R_3 are working
I_9	T_3	Both robots R_3 y R_1 are working
t_1	T_1	Gets its left hand side robot R_1
t_4	T_2	Gets its left hand side robot R_2
t_7	T_3	Gets its left hand side robot R_3
t_2	T_1	Gets its right hand side robot R_2
t_5	T_2	Gets its right hand side robot R_3
t_8	T_3	Gets its right hand side robot R_1
t_3	T_1	Release of robots R_1 y R_2
t_6	T_2	Release of robots R_2 y R_3
t_9	T_3	Release of robots R_3 y R_1

At the beginning all the trasnports and robots are free, the initial marking is $\mu 0 = (1, 0, 0, 1, 0, 0, 1, 0, 0, 1, 1, 1)$. The three transports are equested in concurrent way by their left hand side robots. As a result, transitions t1, t4 and t7 are able to carry onas shown in figurea 6. If t1 is triggered, it means that the controller allowed to t1 to get its left hand side robot 1, the marking will be: $\mu 1 = (0, 1, 0, 1, 0, 0, 1, 0,0, 0, 1, 1)$. This means that T1 has got robot R1 and it is requesting robot R2, in the mean while, the events remain wothout change.

4. SIMULATION SOFTWARE

In this work a simulation package for Petri Nets was developed. The simulator offers an user frendly interface. As shown in figure 6.

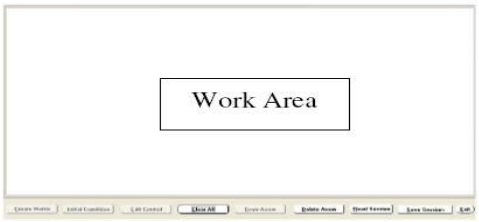


Figure 6

The operator inputs the data related with the PN structure, the information is then interpreted by the simulator which shows the development of the PN.

The following application (shown in figure 7) was introduce to the simulator. The system simulation screen is shown in figure 8.

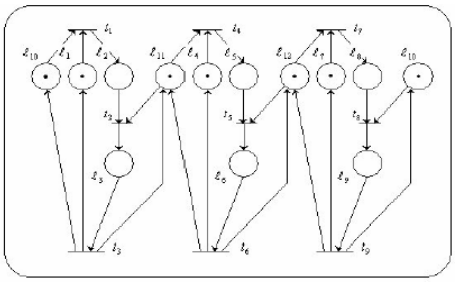


Figure 7. Petri Network for the Flexible Manufacturing Cell

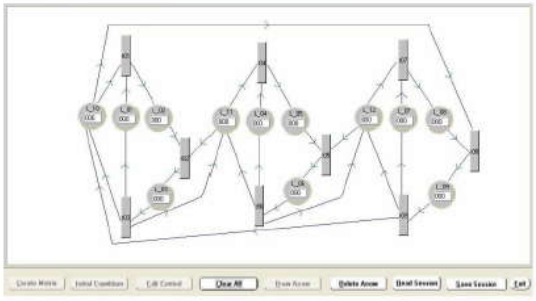


Figure 8. Interface User – Simulator

In figure 8 is also shown the initial marking of the PN. The user can activate a transition by clicking on the boxes (which represent Transitions). On figure 9 & 10 we show the evolution of the PN when some of the transitions are executed.

It is worth to say that this simulator can easily be used as a Control System just by adding an Acquisition Board for sampling the actual signals which in turn can activate the transitions. The simulator also offers the functionality of saving the current status of the simulation in a text file were you can easily edit the parameters. The last means you can change the last status of the net for any other purpose.

Following we present the evolution of the Petri Net through the simulation process.

Figure 9

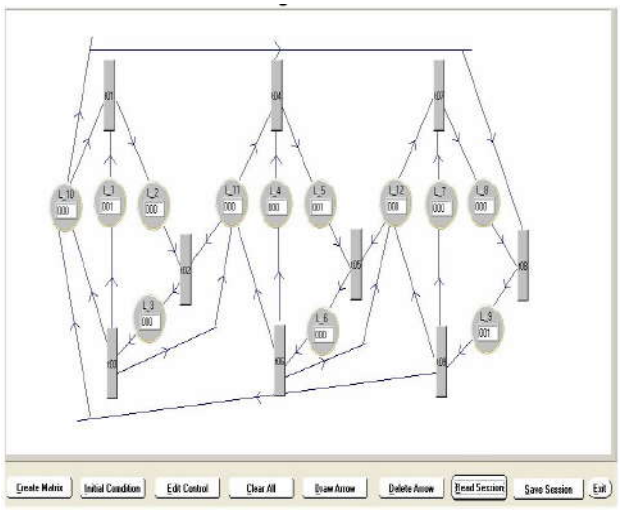
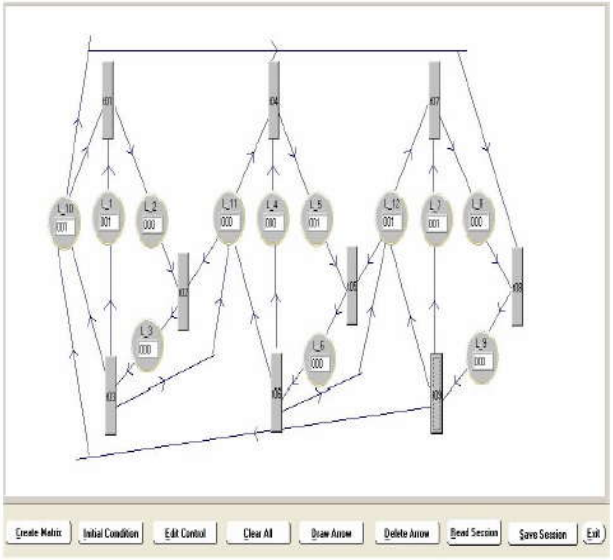


Figure 10 (After transiton #9)



REFERENCIAS

1. Al-Jaar, Y. R., and Desrochers, A. A: "Petri Nets in automation and manufacturing" Robotics and Automation Laboratory, Rensselaer Polytechnic Institute, Troy, New York November, 1982.

2. Brams, G. W.: "Reseaux de Petri, Theorie et Practique" Masson, Paris, 1982. Two volumes.

3. Diaz. M.: "Modeling and Analysis of Communication and Cooperation Protocols Using Petri Net Based Models" North-Holland Publishing Company, Computer Networks, v-6, pp 419-441, 1982.

4. Esteban, P.: "Algorithmes Simplifies d'Analyse des Reseaux de Petri". R. A. I. R. O. APII, Vol. 21, No. 6, 1987.

5. Martínez, J., Alla ; H., and Silva M.: "Petri Nets for the Specification of FMS" Elsevier Science Publishers B. V., pp 389-406, Amsterdam, 1986.

6. Murata, T. Komoda, N. and Matsumoto, K.: "A Petri Net Based Controllers for Flexible and Maintable Secuence Control and its Aplications in Factory Automation". IEEE Transaction on Industrial Electronics, pp 1-8, Vol IE-33, No.1, 1986.

7. Tellez-Giron. R.: "Notas para el Curso de Circuitos Lógicos". Departamento de Ing. Eléctrica CINVESTAV, INP, Vol. II, 1979.

APÉNDICE A.

Código VHDL del bloque lógico de la frecuencia de muestreo.

Listado 1.1 del código VHDL de la frecuencia de muestreo. Módulo principal de este bloque.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity Sampling is
	port(
	RST : in  std_logic;
	CLK : in  std_logic;
	Fs  : out std_logic
	);
end Sampling;
architecture Contador of Sampling is
signal Qn, Qp : std_logic_vector(15 downto 0);
begin
	Combinacional: process(Qp)
	begin
		if (Qp="1100001101001111") then
			Fs <= '1';
			Qn <= (others => '0');
		else
			Fs <= '0';
			Qn <= Qp + 1;
		end if;
	end process Combinacional;

	Secuencial: process(RST,CLK)
	begin
		if (RST='1') then
			Qp <= (others => '0');
		elsif (CLK'event and CLK='1') then
			Qp <= Qn;
		end if;
	end process Secuencial;

end Contador;
```

APÉNDICE B.

Código VHDL del bloque lógico de entrada a la neurona de entrada.

Listado 2.1 del código VHDL de la entrada. Módulo principal de este bloque.

```
-- Registro_entrada
-- Bloque de registros de entrada
-- para el retardo en la muestra x(k)

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;    -- Libreria aritmetica
use IEEE.std_logic_unsigned.all; -- Libreria absoluta

entity Registro_entrada is
        generic(
        n : integer := 18
        );
        port(
        RST       : in  std_logic;
        CLK       : in  std_logic;
        Control   : in  std_logic;
        X1        : out std_logic_vector(n-1 downto 0);
        X2        : out std_logic_vector(n-1 downto 0);
        X3        : out std_logic_vector(n-1 downto 0);
        Listo     : out std_logic
        );
end Registro_entrada;

architecture Completa of Registro_entrada is

component Contador_3
  port(
            RST     : in  std_logic; -- Reset maestro
            CLK     : in  std_logic; -- Reloj maestro
            Fs      : in  std_logic;
            Y       : out std_logic_vector(3 downto 0) -- Cuenta
            );
 end component;
```

```
component LUT
  port(
    Y       : in  std_logic_vector(3 downto 0);
    Xr      : out std_logic_vector(17 downto 0)
    );
  end component;

component Registro_n
        generic(
        n : integer := 18
        );
        port(
        RST     : in  std_logic;
        CLK     : in  std_logic;
        Fs      : in  std_logic;
        Xr      : in  std_logic_vector(n-1 downto 0);
        X1      : out std_logic_vector(n-1 downto 0)
        );
end component;

component corrimiento_FSM
        port(
        RST      : in  std_logic;
        CLK      : in  std_logic;
        Control  : in  std_logic;
        Fs       : out std_logic;
        Listo    : out std_logic
        );
end component;

signal Y :std_logic_vector(3 downto 0);
signal XK1, XK2,Xr : std_logic_vector(17 downto 0);
signal Fs : std_logic;
begin

        X1 <= XK1;

        X2 <= XK2;
        Registro_01: Contador_3     port map(RST,CLK,Fs,Y);
        Registro_02: LUT            port map(Y,Xr);
        Registro_03: Registro_n     port map(RST,CLK,Fs,Xr,XK1);
        Registro_04: Registro_n     port map(RST,CLK,Fs,XK1,XK2);
        Registro_05: Registro_n     port map(RST,CLK,Fs,XK2,X3);
        Registro_06: corrimiento_FSM port map(RST,CLK,Control,Fs,Listo);
```

```
end Completa;
```

Listado 2.2 del código VHDL del componente del contador con habilitación.

```
library IEEE;
use IEEE.std_logic_1164.all;     -- Libreria estandar
use IEEE.std_logic_arith.all;    -- Libreria aritmetica
use IEEE.std_logic_unsigned.all; -- Libreria absoluta
entity Contador_3 is
            port(
      RST     : in  std_logic; -- Reset maestro
      CLK     : in  std_logic; -- Reloj maestro
      Fs      : in  std_logic; -- Habilitacion
      Y       : out std_logic_vector(3 downto 0) -- Cuenta
               );
end Contador_3;

architecture Habilitacion of Contador_3 is
signal Qn, Qp : std_logic_vector(3 downto 0); -- Estados de la maquina
          begin
Combinacional: process(Qp,Fs)
    begin
      if (Fs='0') then
         Qn <= Qp; -- Conservar el estado actual
     else
         Qn <= Qp + 1; -- Decrementar la cuenta actual
             end if;
                Y  <= Qp;     -- Salida
end process Combinacional;
Secuencial: process(RST,CLK)
   begin
      if (RST='1') then
         Qp <= (others => '0'); -- Iniciar en cero
      elsif (CLK'event and CLK='1') then
         Qp <= Qn;
      end if;
   end process Secuencial;
end Habilitacion;
```

Listado 2.3 del código VHDL del componente de la señal de referencia.

```
-- Tabla exhaustiva de equivalencias de fase a funcion normalizada
-- Ajuste de funciones trascendentales

library IEEE;
use IEEE.std_logic_1164.all;

entity LUT is
  port(
    Y : in  std_logic_vector(3 downto 0);
    Xr : out std_logic_vector(17 downto 0)
    );
  end LUT;

architecture Tabla of LUT is
signal I:std_logic_vector(17 downto 0);
begin
  process(Y)
  begin
    case Y is
      when "0000" => I <= "000101011101010001"; -- Indice 0    Valor de la funcion
0.50000000
      when "0001" => I <= "001011110111111010"; -- Indice 1    Valor de la funcion
0.50154114
      when "0010" => I <= "000000001110110000"; -- Indice 2    Valor de la funcion
0.50306702
      when "0011" => I <= "001000000100101110"; -- Indice 3    Valor de la funcion
0.50460815
      when "0100" => I <= "001000000110010010"; -- Indice 4    Valor de la funcion
0.50613403
      when "0101" => I <= "001000000111110111"; -- Indice 5    Valor de la funcion
0.50767517
      when "0110" => I <= "001000001001011011"; -- Indice 6    Valor de la funcion
0.50920105
      when "0111" => I <= "001000001011000000"; -- Indice 7    Valor de la funcion
0.51074219
      when "1000" => I <= "001000001100100100"; -- Indice 8    Valor de la funcion
0.51226807
      when "1001" => I <= "001000001110001001"; -- Indice 9    Valor de la funcion
0.51380920
      when "1010" => I <= "001000001111101101"; -- Indice 10   Valor de la funcion
0.51533508
      when "1011" => I <= "001000010001010001"; -- Indice 11   Valor de la funcion
0.51686096
```

```
      when "1100" => I <= "0010000100101101l0"; -- Indice 12   Valor de la funcion
0.51840210
      when "1101" => I <= "0010000101000110l0"; -- Indice 13   Valor de la funcion
0.51992798
      when "1110" => I <= "0010000101011111l1"; -- Indice 14   Valor de la funcion
0.52146912
      when others => I <= "0010000101111000l1"; -- Indice 15   Valor de la funcion
0.52299500

      -- when others => null;
    end case;
  end process;
  Xr <= I ;
end Tabla;
```

Listado 2.4 del código VHDL del componente de la señal de referencia.

```
-- Registro_n
-- Registro generico individual

library IEEE;
use IEEE.std_logic_1164.all;

entity Registro_n is
        generic(
        n : integer := 18
        );
        port(
        RST    : in  std_logic;
        CLK    : in  std_logic;
        Fs     : in  std_logic;
        Xr     : in  std_logic_vector(n-1 downto 0);
        X1     : out std_logic_vector(n-1 downto 0)
        );
end Registro_n;

architecture Habilitacion of Registro_n is

signal Qn, Qp : std_logic_vector(n-1 downto 0);

begin

        Combinacional: process(Fs,Xr,Qp)
        begin
```

```
                if (Fs='1') then
                        Qn <= Xr;
                else
                        Qn <= Qp;
                end if;
                X1 <= Qp;
        end process Combinacional;

        Secuencial: process(RST,CLK)
        begin
                if (RST='1') then
                        Qp <= (others => '0');
                elsif (CLK'event and CLK='1') then
                        Qp <= Qn;
                end if;
        end process Secuencial;

end Habilitacion;
```

Listado 2.5 del código VHDL de la máquina de control.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity corrimiento_FSM is
        port(
        RST       : in  std_logic;
        CLK       : in  std_logic;
        Control   : in  std_logic;
        Fs        : out std_logic;
        Listo     : out std_logic
        );
end corrimiento_FSM;

architecture FSM of corrimiento_FSM is
signal Qp, Qn : std_logic_vector(2 downto 0);
begin
        Combinacional:process(Control,Qp)
        begin

                case Qp is
                when "000" =>
                if (Control ='0')then
                        Qn<=Qp;
                else
                        Qn <= "001";
```

```
                end if;

                Fs      <= '0';
                Listo   <= '1';

                when "001" =>
                Qn <= "010";

                Fs      <= '1';
                Listo   <= '0';

          when "010" =>
                        if(Control='1')then
                   Qn <= Qp;
                        else
                        Qn <= "000";
                        end if;
                Fs      <= '0';
                Listo   <= '1';

                when others =>
                   Qn <= "000";
                Fs      <= '0';
                Listo   <= '1';


        end case;
end process Combinacional;
Secuencial:process(RST,CLK)
        begin
                if(RST = '1')then
                        Qp <= (others => '0');
                elsif(CLK' event and CLK ='1')then
                        Qp <= Qn;
                end if;
        end process Secuencial;
end FSM;
```

APÉNDICE C.

Código VHDL del bloque lógico de Multiplicador acumulador de la entrada a la neurona de la capa escondida.

Listado 3.1 del código VHDL del multiplicador acumulador de la entrada a la neurona de la capa escondida. Módulo principal de este bloque.

```
--Calculo de los valores de entrada de la neurona de la capa escondida
-- los valores son referidos como la sumatoria de Sji

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity capa_escondida is
        generic (
        n:integer:=18
        );
        port(
        RST         : in std_logic;
        CLK         : in std_logic;
        Control_0   : in  std_logic;
        X1          : in std_logic_vector(17 downto 0);
        X2          : in std_logic_vector(17 downto 0);
        X3          : in std_logic_vector(17 downto 0);
        W11         : in std_logic_vector(17 downto 0);
        W12         : in std_logic_vector(17 downto 0);
        W13         : in std_logic_vector(17 downto 0);
        W21         : in std_logic_vector(17 downto 0);
        W22         : in std_logic_vector(17 downto 0);
        W23         : in std_logic_vector(17 downto 0);
        W31         : in std_logic_vector(17 downto 0);
        W32         : in std_logic_vector(17 downto 0);
        W33         : in std_logic_vector(17 downto 0);
        S1          : out std_logic_vector(n-1 downto 0);
        S2          : out std_logic_vector(n-1 downto 0);
        S3          : out std_logic_vector(n-1 downto 0);
        Listo_0     : out std_logic
        );
end capa_escondida;
architecture network of capa_escondida is
```

```
Component multi_4_1
        port(
        X1      : in std_logic_vector(17 downto 0);
        X2      : in std_logic_vector(17 downto 0);
        X3      : in std_logic_vector(17 downto 0);
        CMX     : in std_logic_vector(1 downto 0);
        X       : out std_logic_vector(17 downto 0)
        );
end Component;

component multi_16_1
        port(
        W11     : in std_logic_vector(17 downto 0);
        W12     : in std_logic_vector(17 downto 0);
        W13     : in std_logic_vector(17 downto 0);
        W21     : in std_logic_vector(17 downto 0);
        W22     : in std_logic_vector(17 downto 0);
        W23     : in std_logic_vector(17 downto 0);
        W31     : in std_logic_vector(17 downto 0);
        W32     : in std_logic_vector(17 downto 0);
        W33     : in std_logic_vector(17 downto 0);
        CMW     : in std_logic_vector(3 downto 0);
        W       : out std_logic_vector(17 downto 0)
        );
end component;

component mult_nbits
        generic (
        n:integer:=18
        );
         port(
        X       : in std_logic_vector(n-1 downto 0);
        W       : in std_logic_vector(n-1 downto 0);
        R1      : out std_logic_vector(2*n-1 downto 0)
        );
end component;

component sumador
        generic (
        n:integer:=18
        );
        port(
        guardar   : in std_logic_vector(n-1 downto 0);
        R1        : in std_logic_vector(n-1 downto 0);
        Suma      : out std_logic_vector(n-1 downto 0)
        ); end component;
```

```
component acumulador
        generic (
        n:integer:=18
        );
        port(
        RST       : in std_logic;
        CLK       : in std_logic;
        LDA       : in std_logic;
        Suma      : in std_logic_vector(n-1 downto 0);
        guardar   : out std_logic_vector(n-1 downto 0)
        );
end component;

component Registro1_n
        generic (
        n:integer:=18
        );
        port(
        RST     : in  std_logic;
        CLK     : in  std_logic;
        LDR     : in  std_logic;
        Suma    : in  std_logic_vector(n-1 downto 0);
        S1      : out std_logic_vector(n-1 downto 0)
        );
end component;

component Registro2_n
        generic (
        n:integer:=18
        );
        port(
        RST      : in  std_logic;
        CLK      : in  std_logic;
        LDR1     : in  std_logic;
        Suma     : in  std_logic_vector(n-1 downto 0);
        S2       : out std_logic_vector(n-1 downto 0)
        );
end component;

component Registro3_n
        generic (
        n:integer:=18
        );
        port(
        RST      : in  std_logic;
```

```
        CLK     : in  std_logic;
        LDR2    : in  std_logic;
        Suma    : in  std_logic_vector(n-1 downto 0);
        S3      : out std_logic_vector(n-1 downto 0)
        ); end component;

component control_FSM
        port(
        RST         : in  std_logic;
        CLK         : in  std_logic;
        Control_0   : in  std_logic;
        CMX         : out  std_logic_vector(1 downto 0);
        CMW         : out  std_logic_vector(3 downto 0);
        LDA         : out std_logic;
        LDR         : out std_logic;
        LDR1        : out std_logic;
        LDR2        : out std_logic;
        Listo_0     : out std_logic
        );
end component;

signal S,W,X : std_logic_vector(n-1 downto 0);
signal R : std_logic_vector(2*n-1 downto 0);
signal R1, guardar,suma: std_logic_vector(2*n+1 downto 0);
signal LDA,LDR,LDR1,LDR2 :std_logic;
signal CMX : std_logic_vector(1 downto 0);
signal CMW : std_logic_vector(3 downto 0);

begin
        archivo_1: multi_4_1 port map(X1,X2,X3,CMX,X);
        archivo_2: multi_16_1 port map
        (W11,W12,W13,W21,W22,W23,W31,W32,W33,CMW,W);
        archivo_3: mult_nbits generic map(n) port map(X,W,R);

        R1 <= R(2*n-1) & R(2*n-1) & R;

        archivo_4: sumador generic map(2*n+2) port map(guardar,R1,Suma);

        archivo_5: acumulador generic map(2*n+2) port map
        (RST,CLK,LDA,Suma,guardar);

        S <= Suma(33 downto 16);

        archivo_6: Registro1_n generic map(n) port map(RST,CLK,LDR,S,S1);
        archivo_7: Registro2_n generic map(n) port map(RST,CLK,LDR1,S,S2);
        archivo_8: Registro3_n generic map(n) port map(RST,CLK,LDR2,S,S3);
```

```
        archivo_9: control_FSM port map (RST,CLK, Control_0,CMX, CMW, LDA,
        LDR, LDR1, LDR2,Listo_0);

end network;
```

Listado 3.2 del código VHDL del componente del multiplexor 4_1.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity multi_4_1 is
        port(
        X1      : in std_logic_vector(17 downto 0);
        X2      : in std_logic_vector(17 downto 0);
        X3      : in std_logic_vector(17 downto 0);
        CMX     : in std_logic_vector(1 downto 0);
        X       : out std_logic_vector(17 downto 0)
        );
end multi_4_1;

architecture simple_1 of multi_4_1 is
begin
        process(X1,X2,X3,CMX)
        begin
                case CMX is
                        when "00"   => X <= X1;
                        when "01"   => X <= X2;
                        when "10"   => X <= X3;
                        when others => X <= (others => '0');
                end case;
        end process;
end simple_1;
```

Listado 3.3 del código VHDL del componente del multiplexor 16_1.

```
library IEEE;
use IEEE.std_logic_1164.all;
entity multi_16_1 is
        port(
        W11     : in std_logic_vector(17 downto 0);
        W12     : in std_logic_vector(17 downto 0);
        W13     : in std_logic_vector(17 downto 0);
        W21     : in std_logic_vector(17 downto 0);
        W22     : in std_logic_vector(17 downto 0);
        W23     : in std_logic_vector(17 downto 0);
```

```
        W31     : in std_logic_vector(17 downto 0);
        W32     : in std_logic_vector(17 downto 0);
        W33     : in std_logic_vector(17 downto 0);
        CMW     : in std_logic_vector(3 downto 0);
        W       : out std_logic_vector(17 downto 0)
        );
end multi_16_1;

architecture simple_2 of multi_16_1 is
begin
        process(W11,W12,W13,W21,W22,W23,W31,W32,W33,CMW)
        begin
                case CMW is
                        when "0000"   => W <= W11;
                        when "0001"   => W <= W12;
                        when "0010"   => W <= W13;
                        when "0011"   => W <= W21;
                        when "0100"   => W <= W22;
                        when "0101"   => W <= W23;
                        when "0110"   => W <= W31;
                        when "0111"   => W <= W32;
                        when "1000"   => W <= W33;
                        when others => W <= (others => '0');
                end case;
        end process;
end simple_2;
```

Listado 3.4 del código VHDL del componente del mult_nbits.

```
Library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;
entity mult_nbits is
        generic(
        n:integer:=18
        );
   port(
        X    : in std_logic_vector(n-1 downto 0);
        W    : in std_logic_vector(n-1 downto 0);
        R1   : out std_logic_vector(2*n-1 downto 0)
        );
end mult_nbits;

architecture completa_1 of mult_nbits is
begin
```

```
        process(X,W)
        variable M:signed(2*n-1 downto 0);
        begin
                M:=signed(X)*signed(W);
                R1 <= std_logic_vector(M);
        end process;
end completa_1;
```

Listado 3.5 del código VHDL del componente del sumador.

```
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity sumador is
        generic (
        n:integer:=18
        );
        port(
        guarder    : in std_logic_vector(n-1 downto 0);
        R1         : in std_logic_vector(n-1 downto 0);
        Suma       : out std_logic_vector(n-1 downto 0)
        );
end sumador;

architecture red of sumador is
begin
        process(guardar,R1)
        variable S: unsigned(n downto 0);
        begin
        S:= unsigned('0'& guardar) + unsigned('0' & R1);
        Suma <= std_logic_vector(S(n-1 downto 0));
        end process;

end red;
```

Listado 3.6 del código VHDL del componente del acumulador.

```
--acumulador

library IEEE;
use IEEE.std_logic_1164.all;
entity acumulador is
        generic(
        n:integer:=18
```

```
        );
        port(
        RST    : in std_logic;
        CLK    : in std_logic;
        LDA    : in std_logic;
        Suma    : in std_logic_vector(n-1 downto 0);
        guardar : out std_logic_vector(n-1 downto 0)
        );
end acumulador;
architecture Sumandos of acumulador is
signal Ap,An :std_logic_vector(n-1 downto 0);
begin
        Combinacional:process(Suma,LDA,Ap)
        begin
        if(LDA ='0')then
                An <= (others => '0');
        else
                An <= Suma;
        end if;
           guardar <= Ap;
        end process Combinacional;
        Secuencial:process(RST,CLK)
        begin
                if(RST = '1')then
                        Ap <= (others => '0');
                elsif(CLK' event and CLK ='1')then
                        Ap <= An;
                end if;
        end process Secuencial;
end sumandos;
```

Listado 3.7 del código VHDL del componente del registro de resultado uno.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Registro1_n is
        generic(
        n : integer := 18
        );
        port(
        RST     : in  std_logic;
        CLK     : in  std_logic;
        LDR     : in  std_logic;
        Suma    : in  std_logic_vector(n-1 downto 0);
```

```
        S1        : out std_logic_vector(n-1 downto 0)
        );
end Registro1_n;

architecture habilitada of Registro1_n is
signal Qn, Qp : std_logic_vector(n-1 downto 0);
begin
        Combinacional: process(LDR,Suma,Qp)
        begin
                if (LDR='0') then
                        Qn <= Qp;
                else
                        Qn <= Suma;
                end if;
                S1 <= Qp;
        end process Combinacional;

        Secuencial: process(RST,CLK)
        begin
                if (RST='1') then
                        Qp <= (others => '0');
                elsif (CLK'event and CLK='1') then
                        Qp <= Qn;
                end if;
        end process Secuencial;
end Habilitada;
```

Listado 3.8 del código VHDL del componente del registro de resultado dos.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Registro2_n is
        generic(
        n : integer := 18
        );
        port(
        RST      : in  std_logic;
        CLK      : in  std_logic;
        LDR1     : in  std_logic;
        Suma     : in  std_logic_vector(n-1 downto 0);
        S2       : out std_logic_vector(n-1 downto 0)
        );
end Registro2_n;
```

```
architecture habilitada_1 of Registro2_n is
signal Qn, Qp : std_logic_vector(n-1 downto 0);
begin
	Combinacional: process(LDR1,Suma,Qp)
	begin
		if (LDR1='0') then
			Qn <= Qp;
		else
			Qn <= Suma;
		end if;
	      S2 <= Qp;
	end process Combinacional;

	Secuencial: process(RST,CLK)
	begin
		if (RST='1') then
			Qp <= (others => '0');
		elsif (CLK'event and CLK='1') then
			Qp <= Qn;
		end if;
	end process Secuencial;
end Habilitada_1;
```

Listado 3.9 del código VHDL del componente del registro de resultado tres.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity Registro3_n is
	generic(
	n : integer := 18
	);
	port(
	RST	: in  std_logic;
	CLK	: in  std_logic;
	LDR2	: in  std_logic;
	Suma	: in  std_logic_vector(n-1 downto 0);
	S3	: out std_logic_vector(n-1 downto 0)
	);
end Registro3_n;

architecture habilitada_2 of Registro3_n is
signal Qn, Qp : std_logic_vector(n-1 downto 0);
begin
```

```
	Combinacional: process(LDR2,Suma,Qp)
	begin
		if (LDR2='0') then
			Qn <= Qp;
		else
			Qn <= Suma;
		end if;
		S3 <= Qp;
	end process Combinacional;

	Secuencial: process(RST,CLK)
	begin
		if (RST='1') then
			Qp <= (others => '0');
		elsif (CLK'event and CLK='1') then
			Qp <= Qn;
		end if;
	end process Secuencial;
end Habilitada_2;
```

Listado 3.10 del código VHDL del componente de la máquina de control de las operaciones.

```
library IEEE;
use IEEE.std_logic_1164.all;

entity control_FSM is
	port(
	RST		: in  std_logic;
	CLK		: in  std_logic;
	Control_0	: in  std_logic;
	CMX		: out  std_logic_vector(1 downto 0);
	CMW		: out std_logic_vector(3 downto 0);
	LDA		: out std_logic;
	LDR		: out std_logic;
	LDR1		: out std_logic;
	LDR2		: out std_logic;
	Listo_0		: out std_logic
	);
end control_FSM;

architecture FSM of control_FSM is
signal Qp, Qn : std_logic_vector(3 downto 0);
begin
	Combinacional:process(control_0,Qp)
```

```
begin
  --primera multiplicacion     S1
        case Qp is
        when "0000" =>
        if (control_0 ='0')then
                Qn<=Qp;
        else
                Qn <= "0001";
        end if;
        -- entradas x
        CMX(1) <= '1';
        CMX(0) <= '1';
        -- entradas w
        CMW(3) <= '1';
        CMW(2) <= '1';
        CMW(1) <= '1';
        CMW(0) <= '1';

        LDA      <= '0';
        LDR      <= '0';
        LDR1     <= '0';
        LDR2     <= '0';
        listo_0    <= '1';

        when "0001" =>
        Qn <= "0010";
        -- entradas x
        CMX(1) <= '0';
        CMX(0) <= '0';
        -- entradas w
        CMW(3) <= '0';
        CMW(2) <= '0';
        CMW(1) <= '0';
        CMW(0) <= '0';

        LDA      <= '1';
        LDR      <= '0';
        LDR1     <= '0';
        LDR2     <= '0';
        listo_0    <= '0';

        when "0010" =>
        Qn <= "0011";
        -- entradas x
        CMX(1) <= '0';
        CMX(0) <= '1';
```

```
        -- entradas w
        CMW(3) <= '0';
        CMW(2) <= '0';
        CMW(1) <= '1';
        CMW(0) <= '1';

        LDA     <= '1';
        LDR     <= '0';
        LDR1    <= '0';
        LDR2    <= '0';
        listo_0 <= '0';

        when "0011" =>
        Qn <= "0100";
        -- entradas x
        CMX(1) <= '1';
        CMX(0) <= '0';
        --entradas w
        CMW(3) <= '0';
        CMW(2) <= '1';
        CMW(1) <= '1';
        CMW(0) <= '0';

        LDA     <= '0';
        LDR     <= '1';
        LDR1    <= '0';
        LDR2    <= '0';
        listo_0 <= '0';

--segunda multiplicacion S2

when "0100" =>
        Qn <= "0101";
        --entradas x
        CMX(1) <= '0';
        CMX(0) <= '0';
        -- entradas w
        CMW(3) <= '0';
        CMW(2) <= '0';
        CMW(1) <= '0';
        CMW(0) <= '1';

        LDA     <= '1';
        LDR     <= '0';
        LDR1    <= '0';
        LDR2    <= '0';
```

```
        listo_0    <= '0';

        when "0101" =>
        Qn <= "0110";
        -- entradas x
        CMX(1) <= '0';
        CMX(0) <= '1';
        -- entradas w
        CMW(3) <= '0';
        CMW(2) <= '1';
        CMW(1) <= '0';
        CMW(0) <= '0';

        LDA      <= '1';
        LDR      <= '0';
        LDR1     <= '0';
        LDR2     <= '0';
        listo_0    <= '0';

        when "0110" =>
        Qn <= "0111";
        -- entradas x
        CMX(1) <= '1';
        CMX(0) <= '0';
        -- entradas w
        CMW(3) <= '0';
        CMW(2) <= '1';
        CMW(1) <= '1';
        CMW(0) <= '1';

        LDA      <= '0';
        LDR      <= '0';
        LDR1     <= '1';
        LDR2     <= '0';
        listo_0    <= '0';

--tercera multiplicación S3


        when "0111" =>
        Qn <= "1000";
        -- entradas x
        CMX(1) <= '0';
        CMX(0) <= '0';
        -- entradas w
        CMW(3) <= '0';
```

```
        CMW(2) <= '0';
        CMW(1) <= '1';
        CMW(0) <= '0';

        LDA     <= '1';
        LDR     <= '0';
        LDR1    <= '0';
        LDR2    <= '0';
        listo_0   <= '0';

        when "1000" =>
        Qn <= "1001";
        -- entradas x
        CMX(1) <= '0';
        CMX(0) <= '1';
        -- entradas w
        CMW(3) <= '0';
        CMW(2) <= '1';
        CMW(1) <= '0';
        CMW(0) <= '1';

        LDA     <= '1';
        LDR     <= '0';
        LDR1    <= '0';
        LDR2    <= '0';
        listo_0   <= '0';

        when "1001" =>
        Qn <= "1010";
        -- entradas x
        CMX(1) <= '1';
        CMX(0) <= '0';
        --entradas w
        CMW(3) <= '1';
        CMW(2) <= '0';
        CMW(1) <= '0';
        CMW(0) <= '0';

        LDA     <= '0';
        LDR     <= '0';
        LDR1    <= '0';
        LDR2    <= '1';
        listo_0   <= '0';

when "1010" =>
                if(control_0='1')then
```

```
                    Qn <= Qp;
                        else
                        Qn <= "0000";
                        end if;
                    CMX(1) <= '1';
                    CMX(0) <= '1';
                    CMW(3) <= '1';
                    CMW(2) <= '1';
                    CMW(1) <= '1';
                    CMW(0) <= '1';

                    LDA     <= '0';
                    LDR     <= '0';
                    LDR1    <= '0';
                    LDR2    <= '0';
                    listo_0 <= '1';

                    when others =>
                    Qn <= "0000";

                    CMX(1) <= '1';
                    CMX(0) <= '1';
                    CMW(3) <='1';
                    CMW(2) <= '1';
                    CMW(1) <='1';
                    CMW(0) <='1';

                    LDA     <= '0';
                    LDR     <=  '0';
                    LDR1    <= '0';
                    LDR2    <= '0';
                    listo_0 <= '1';
        end case;
end process Combinacional;
Secuencial:process(RST,CLK)
        begin
                if(RST = '1')then
                        Qp <= (others => '0');
                elsif(CLK' event and CLK ='1')then
                        Qp <= Qn;
                end if;
        end process Secuencial;
end FSM;
```

ANEXOS

CONCLUSIONES

En este trabajo los resultados para la implementación de la red neuronal artificial Autoajustable no se pudieron llevar acabo porque la descripción del algoritmo de aprendizaje de Retropropagación hacia atrás de la red en VHDL es demasiado grande, ocasionando que el FPGA seleccionado para la implementación de este algoritmo, no tuviera los recursos necesarios para llevar cabo ésta, dejando como base para futuras investigaciones la implementación de controles inteligentes modelado por redes neuronales artificiales, saber seleccionar el hardware(FPGA) adecuado para la implementación de las redes neuronales artificiales, sin embargo, se dejan los recursos necesarios para conseguir el objetivo de este trabajo, de acuerdo a los avances de la tecnología, se puede encontrar más adelante el hardware (FPGA) con más recursos para poder solucionar el problema, a un costo no muy elevado, para así justificar los alcances de este proyecto.

También nos queda la experiencia de diseños demasiados grandes, podemos ver el alcance a un costo beneficio, donde podemos concluir si es factible o no factible su implementación en un solo circuito integrado. (SOC), se deja la investigación realizada y la descripción del algoritmo de aprendizaje de la red neuronal artificial Autoajustable para obtener los resultados requeridos que se plantearon a un inicio de este trabajo, donde el objetivo era la implementación de un control inteligente modelado por una red neuronal artificial Autoajustable para controlar un servo motor de corriente directa.

La parte de la simulación de la descripción del algoritmo de aprendizaje de la red neuronal artificial Autoajustable, los resultados obtenidos fueron satisfactorios, los cuales fueron comprobados en todos los bloques lógicos que describieron los pasos del algoritmo de aprendizaje de Retropropagación hacia atrás de la red. Cada bloque se reviso en el software de síntesis Xilinx ISE 8.2j obteniendo los

bloque digitales de cada paso del algoritmo de aprendizaje, que se plantearon en la metodología de este trabajo.