CENTRO DE INVESTIGACIÓN Y DE ESTUDIOS AVANZADOS DEL IPN

UNIDAD ZACATENCO

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

SECCIÓN DE ELECTRÓNICA DEL ESTADO SÓLIDO

DESARROLLO EN FPGA DE UN EMULADOR DE PANEL FOTOVOLTAICO

TESIS

Que presenta

ING. GERARDO MARCOS TORNEZ XAVIER

Para obtener el grado de

MAESTRO EN CIENCIAS

EN LA ESPECIALIDAD DE INGENIERÍA ELÉCTRICA

Directores de Tesis:

DR. FELIPE GÓMEZ CASTAÑEDA

DR. JOSÉ ANTONIO MORENO CADENAS

México D.F. Enero 2014

Dedicatorias

Gracias Dios por siempre darme las fuerzas para continuar.

Este trabajo se lo dedico: al Sr. Gerardo Tornez Garibo y Sra. Osmar Xavier Pastrana, por el gran apoyo y cariño que me han brindado a lo largo de mi vida, ya que sin escatimar esfuerzo alguno han sacrificado gran parte de su vida para formarme y educarme. A ellos va con todo mi amor y cariño por ser los mejores padres del mundo.

A mi pequeña hermana, a quien quiero y aprecio: Geraldine Esmeralda Tornez Xavier, por todas sus risas y apoyo brindado en los momentos más difíciles.

Agradecimientos

A mis asesores de tesis:

Dr. Felipe Gómez Castañeda y Dr. José Antonio Moreno Cadenas, por su valiosa guía y enseñanza brindada durante mi formación educativa y a lo largo de la realización de esta tesis.

A mis revisores y sinodales:

Aprecio y agradezco el tiempo dedicado a la revisión de esta tesis al Dr. Alfredo Reyes Barranca (CINVESTAV - SEES) y Dr. Mariano Gamboa Zúñiga (CINVESTAV - CGSTIC) ya que sus consejos y sugerencias me fueron de gran ayuda para el enriquecimiento de este trabajo.

A mis compañeros del CINVESTAV

Al M. en C. Luis Martín Flores Nava por sus consejos y apoyo brindado durante todo el desarrollo de este trabajo, al Dr. Oliverio Arellano Cárdenas, Ing. Emilio Espinosa García, por su diario quehacer en el laboratorio de VLSI.

Al M. en C. Omar Hernández Garnica y M. en C. José Luis Ochoa Padilla, por sus consejos brindados.

Agradezco al Consejo Nacional de Ciencia y Tecnología (CONACYT) por la beca otorgada para la realización de éste trabajo.

Agradezco al CINVESTAV, por la educación recibida.

Índice.

Capítulo 3: Implementación de una red neuronal en Matlab.

Capítulo 4: Implementación de una red neuronal en lenguaje VHDL.

Capítulo 5: Resultados generales.

Capítulo 6: Conclusiones.

Resumen.

Esta tesis aborda el diseño de un emulador electrónico que permita aproximar la respuesta eléctrica, que se obtendría al excitar con un cierto valor de irradiancia y temperatura a un módulo solar comercial, por lo cual en la primera parte de este trabajo, se desarrollará el modelo análogo – eléctrico del módulo solar ST5 de la compañía Siemens, con el objetivo de obtener su respuesta eléctrica. Este modelo se simulará utilizando código Spice; también se propondrá la creación de una red neuronal artificial en el ambiente de desarrollo de Matlab, la cual será entrenada utilizando el algoritmo de Levenberg-Marquardt, para reproducir el perfil de salida que se obtendría del módulo solar utilizado. Finalmente, se presentará la implementación de esta red neuronal artificial en un dispositivo digital de Arreglos de Compuertas Programables en Campo (FPGA), el cual cumplirá la función del emulador y nos permitirá realizar simulaciones en tiempo real, como si se tuviera físicamente a un módulo solar, para de esta manera mostrar otra opción más económica, para el modelado de un módulo solar.

Abstract.

This thesis addresses the design of an emulator to approximate the electrical response that would be obtained by exciting a commercial solar module with a certain value of irradiance and temperature , so in the first part of this work, we develop the analogous model - Solar module power company Siemens ST5 , in order to obtain their electrical response; this model will be simulated using a Spice code. Also, this work undertakes the creation of an artificial neural network in a Matlab development environment , which will be trained using the Levenberg - Marquardt algorithm to track the output profile, which would generate the actual solar module. Finally, we present the implementation of this artificial neural network on a digital device FPGA, which will act as an emulator and this allow us to perform simulations in real time, as if the solar module were physically present, and in this way to show another option for the economic modeling of a solar module.

Introducción General.

Hoy en día existe un gran interés por las fuentes de energía renovables ya que se estima que en un futuro exista una disminución en el consumo de las energías fósiles. Una fuente de energía renovable que ha despertado gran interés en el mundo industrial y en el de la investigación es la energía solar, debido a que ésta presenta una gran disponibilidad alrededor del mundo, además de ser una energía no contaminante. Para obtener su máximo rendimiento, a lo largo de los años se han ido mejorando los procesos de fabricación de los módulos solares. Asímismo, cada vez se presentan nuevos modelos matemáticos que tratan de modelar el comportamiento de dichos módulos. Este modelado puede llegar a ser un problema debido a que existen demasiados parámetros naturales que pueden intervenir en dichos modelos. Por tal motivo, se propone el uso de una red neuronal artificial ya que este tipo de herramientas nos permiten resolver problemas en los cuales es difícil obtener un modelo matemático.

Objetivo.

El objetivo de este trabajo de tesis es el de desarrollar un emulador a nivel de hardware, que sea capaz de imitar la respuesta eléctrica de un módulo solar comercial, utilizando una red neuronal artificial multicapa tipo Perceptron, como un aproximador de funciones, para después realizar su implementación en un dispositivo digital FPGA (Field Programmable Gate Array).

Organización de la Tesis.

El trabajo presentado se va a componer de 6 capítulos y 4 Apéndices. El orden cronológico de lectura corresponde con la numeración de los capítulos.

En el ***Capítulo 1***, se abordarán algunos conceptos teóricos referentes a las redes neuronales artificiales.

En el ***Capítulo 2***, se emulará el funcionamiento del panel solar ST5 de la compañía Siemens, a partir de la elaboración de un modelo análogo – eléctrico en código Spice, para la obtención de los parámetros eléctricos conocidos como corriente a corto circuito (I_{SC}) y voltaje a circuito abierto (V_{OC}).

En el ***Capítulo 3***, se realizará el diseño y entrenamiento de una red neuronal perceptrón multicapa, utilizando el ambiente de desarrollo de Matlab.

En el ***Capítulo 4***, se presentará la implementación de la red neuronal artificial en un dispositivo digital FPGA, utilizando el lenguaje de descripción de hardware VHDL y se analizará la respuesta de cada módulo que conforma a dicha red.

En el ***Capítulo 5***, se hace una comparación entre la respuesta obtenida del emulador implementado en el FPGA, con la respuesta obtenida del modelo análogo – eléctrico del panel solar ST5, para de esta manera poder proponer mejoras y optimizaciones para un trabajo futuro.

En el ***Capítulo 6***, se mencionaran las conclusiones generales sobre este trabajo.

CAPÍTULO 1

Marco teórico

1.1 Introducción.

En este capítulo se darán algunos conceptos generales sobre las redes neuronales artificiales, como su arquitectura y el algoritmo de aprendizaje utilizado para su entrenamiento, que serán útiles para la implementación de una red neuronal multicapa configurada como aproximador de funciones. Además, se verán algunos conceptos del dispositivo digital FPGA.

1.2 Redes Neuronales Artificiales (RNA).

El desarrollo de las redes neuronales artificiales empezó aproximadamente 50 años atrás y la principal motivación de su desarrollo fue el deseo de entender y emular algunas de las características del sistema nervioso y/o cerebro humano. Por tal motivo, éstas permiten emular características propias de los seres humanos como la capacidad de memorizar y asociar hechos. Una red neuronal artificial es un sistema para el tratamiento de la información cuya unidad básica de procesamiento está inspirada en la célula fundamental del sistema nervioso humano, *la neurona*.

El estudio y la aplicación de las redes neuronales artificiales abarcan un campo interdisciplinario muy extenso, ya que son una poderosa herramienta para el modelado de problemas en los cuales la relación explícita entre ciertas variables no se conoce, además de que permite resolver, por ejemplo, problemas de clasificación, reconocimiento de patrones, aproximación de funciones y predicción.

Algunas de las ventajas que presentan las redes neuronales son las siguientes:

Aprendizaje: Las redes neuronales tienen la habilidad de aprender mediante un proceso denominado aprendizaje, el cual consiste en proporcionar a la red neuronal datos como entrada y a su vez indicarle cuál será la salida esperada.

Auto organización: Una red neuronal tiene la capacidad de crear su propia representación a partir de la información contenida en su interior.

Tolerancia a fallos: Debido a que las redes neuronales almacenan información de forma redundante, ésta puede seguir respondiendo aun si existe un daño parcial en la red.

Flexibilidad: Una red neuronal puede tolerar cambios en la información de entrada como señales de ruido.

Tiempo real: La estructura de una red neuronal es paralela, lo cual permite realizar su implementación en computadoras o dispositivos electrónicos especiales (FPGA), permitiendo la obtención de respuestas en tiempo real.

1.2.1 Inspiración biológica.

Biológicamente nuestro cerebro humano se encuentra compuesto de aproximadamente 10^{11} neuronas y cada neurona a su vez presenta 10^4 conexiones con otras neuronas [1]. Recientemente, científicos han empezado a entender cómo operan las redes neuronales biológicas, descubriendo que básicamente las funciones neuronales biológicas como la memoria, se encuentran almacenadas en las neuronas y especialmente en las conexiones entre ellas [2]. A su vez, los científicos ven el proceso de aprendizaje como la generación de nuevas conexiones entre neuronas o la modificación de conexiones existentes.

El poder de nuestro cerebro radica en la forma en la que trabajan las neuronas ya que estas trabajan de forma paralela [3]. Esto quiere decir que todas las neuronas trabajan al mismo tiempo, esta característica es representativa en las redes neuronales artificiales. Una neurona biológica se encuentra conformada por tres principales componentes, como se puede apreciar en la Fig. (1.1).

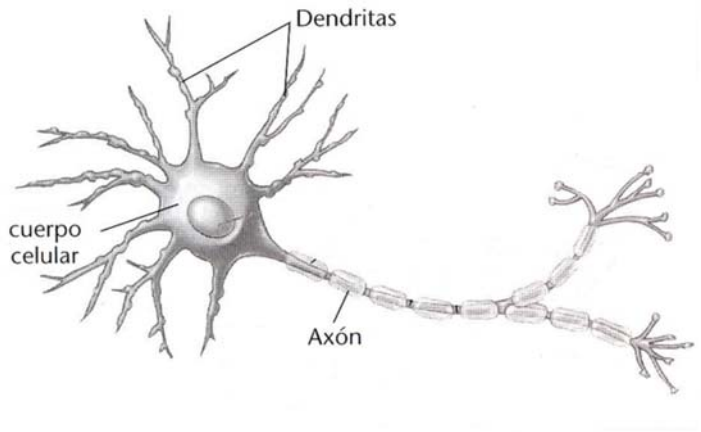


Fig. (1.1) *Diagrama de una neurona biológica.*

Dendritas: Son fibras nerviosas receptivas que se encargan de conducir al interior del cuerpo celular las señales eléctricas.

Cuerpo celular: Aquí se realiza la sumatoria de las señales entrantes con el umbral o polarización.

Axón: Es una larga fibra nerviosa que se encarga de llevar la señal procesada por el cuerpo celular hacia las otras neuronas.

El punto de contacto entre el axón de una neurona y la dendrita de otra neurona se le conoce como *sinapsis*.

Aunque nosotros tenemos un rudimentario entendimiento de las redes neuronales biológicas, es posible realizar la construcción de pequeños conjuntos de simples neuronas artificiales y realizar su entrenamiento para realizar una tarea.

1.2.2 Neurona artificial de una sola entrada.

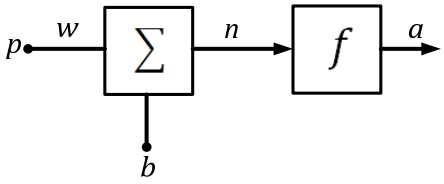


Fig. (1.2) *Representación de una neurona de una entrada.*

A continuación se explicará el funcionamiento básico utilizando la representación de una neurona artificial. En la Fig. (1.2) se muestra el modelo utilizado para representar a una neurona artificial con una sola entrada. Como se puede apreciar, el valor escalar de entrada se representará con la variable (*p*), la cual será multiplicada por un valor escalar denominado por la letra (*w*) el cual servirá para representar a un peso sináptico, para de esta manera formar el producto (*wp*); este producto, como se puede observar, será enviado a un bloque sumador donde se le sumará una polarización o umbral (*b*). El resultado de esta sumatoria recibirá el nombre de *entrada neta* (*n*), la cual será la entrada para el bloque (*f*), utilizado para representar la función de activación o función de transferencia, la cual producirá la salida (*a*) de tipo escalar de la neurona. Este simple modelo puede ser relacionado con el modelo biológico de una neurona de la siguiente manera:

- El peso sináptico, que se representa por la literal *w*, cumplirá la función de realizar la sinapsis biológica de la neurona.
- El cuerpo celular se relacionará con el bloque de la sumatoria.

- El axón quedará representado por la función de transferencia y la salida a.

1.2.3 Modelo matemático simplificado de una neurona artificial.

El modelo matemático que describe el procedimiento anterior, se representa con la ecuación (1.1).

$$a = f(wp + b) \tag{1.1}$$

1.2.4 Red Neuronal Multicapa.

Muchas veces el uso de una neurona no es suficiente para resolver cierto tipo de problemas, como por ejemplo nuestro caso donde se desea resolver un problema de optimización, por lo cual una vez que se entendió el concepto de cómo se encuentra conformada una neurona artificial, se puede usar este concepto para conformar una red neuronal multicapa, donde se pueda involucrar a un mayor número de neuronas en varias capas, operando simultáneamente de forma paralela, como se puede ver en la Fig. (1.3).

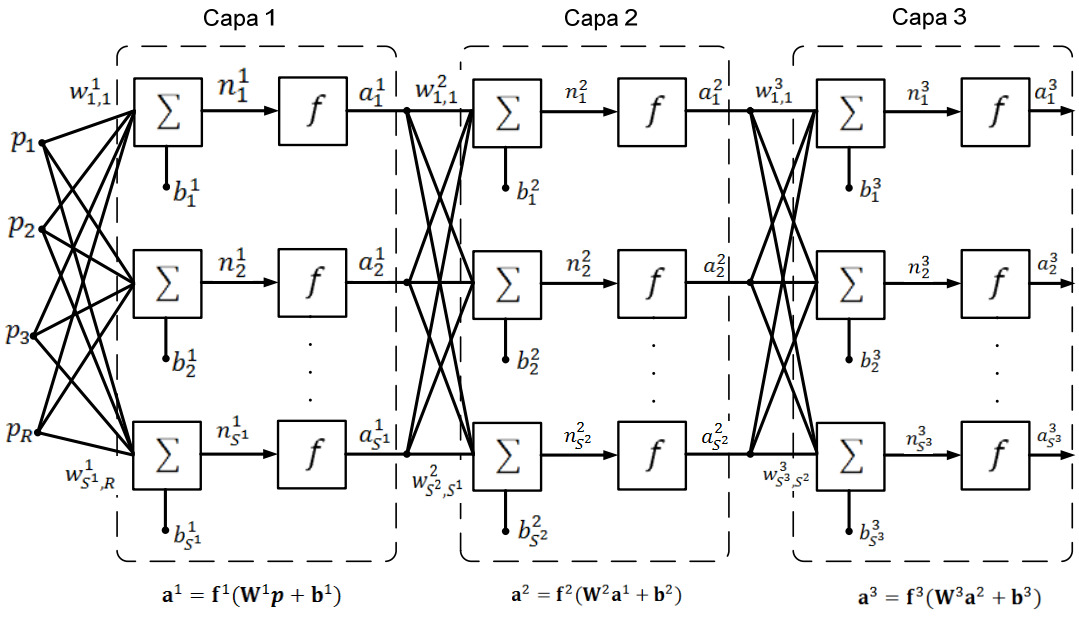


Fig. (1.3) *Diagrama de una red Multicapa.*

Las neuronas que proporcionan la salida de la red neuronal conforman la capa de salida, mientras que las demás neuronas quedan representadas en capas ocultas. Para el ejemplo mostrado en la figura anterior, se tiene una capa de salida (capa 3) y dos capas ocultas (capa 1 y capa 2), además la literal R representará el número total de entradas que se tengan en la red. S será el número de

neurona y el subíndice ubicado en la parte superior de las variables indicará la capa a la cual pertenece dicha variable. Así por ejemplo, el peso sináptico $w_{1,3}^{1}$, conecta a la neurona 1 con la entrada 3, para la capa 1.

1.2.5 Arquitectura.

Otro aspecto importante que se requiere definir, es el tipo de arquitectura de la red neuronal; esta se puede clasificar en dos rubros [4].

- Según el número de capas que presente la red, ya sea unicapa o multicapa.
- El tipo de conexión entre neuronas, para la propagación de los datos.

  Redes Feedforward (hacia adelante).

  Redes Recurrentes.

Nosotros nos enfocaremos en las redes de tipo Feedforward, ya que es el tipo de arquitectura que más se utiliza para la creación de redes multicapa, las cuales son utilizadas para la resolución de problemas referentes al reconocimiento de patrones o a la creación de aproximadores de funciones. Este tipo de arquitectura va a tener la característica de que siempre el flujo de la información desde la entrada hasta la salida siempre será estrictamente hacia adelante, no van a existir retardos ni retroalimentación entre las neuronas de la salida hacia las neuronas de entrada o a alguna neurona contenida en cualquier capa. Esto quiere decir que la salida de las neuronas de una capa se convertirán en las entradas de las neuronas de la siguiente capa. En la Fig. (1.3) se puede apreciar este tipo de arquitectura.

1.2.6 Aprendizaje Supervisado.

En este tipo de aprendizaje, para cada patrón de entrada la red neuronal conocerá el patrón de salida deseado, como se puede ver en la siguiente expresión:

$$\{\mathbf{p}_1, \mathbf{t}_1\}, \{\mathbf{p}_1, \mathbf{t}_1\}, \dots, \{\mathbf{p}_Q, \mathbf{t}_Q\} \tag{1.2}$$

Donde $\mathbf{p}_q$ es una entrada a la red neuronal y $\mathbf{t}_q$ es su correspondiente salida objetivo. Dado que las entradas son aplicadas a la red, las salidas de la red son comparadas con las salidas objetivo, por lo que el proceso de entrenamiento de una red neuronal consiste en ajustar los pesos sinápticos y la polarización de la red, de tal manera que las salidas de la red sean lo más cercano a las salidas objetivo deseadas.

1.2.7 Función de Transferencia.

Las funciones de transferencia o funciones de activación, como también se le suele llamar, que más se utilizan cuando se trabajan con redes multicapas que son entrenadas con el algoritmo de Retro-propagación o con algoritmos que son variantes de la Retro-propagación, son las funciones sigmoidales, ya que se pueden usar para resolver problemas de reconocimiento de patrones o de ajuste de funciones. En este caso nosotros utilizaremos la función denominada Tangente Hiperbólica Sigmoidal, la cual se puede apreciar gráficamente en la Fig. (1.4).

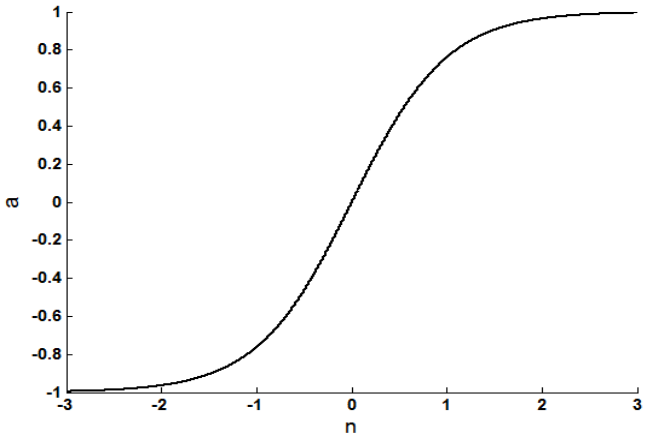


Fig. (1.4) *Función de transferencia Tangente Hiperbólica Sigmoidal.*

Esta función tiene la característica de no ser lineal y de ser diferenciable además de tomar cualquier valor de entrada dentro del rango de +/- ∞ y acotar el valor de la salida dentro de un rango de +/-1; esta función se puede representar con la expresión (1.3).

$$f = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}} \tag{1.3}$$

1.3 Error Cuadrático Medio (MSE).

Este es el índice de desempeño más utilizado cuando se realiza el entrenamiento de una red neuronal multicapa a partir del algoritmo de Retro-propagación o cualquier otro algoritmo que sea variante de éste y está definido por la siguiente ecuación.

$$F(\mathbf{x}) = E[(t-a)^2] \tag{1.4}$$

Lo que se buscará al entrenar una red neuronal es encontrar los valores de los pesos sinápticos y la polarización, de tal manera que la diferencia entre la salida objetivo (t) y el valor obtenido como salida de la red (a), sea mínima.

1.4 Algoritmo de aprendizaje Levenberg-Marquardt.

Cuando el algoritmo básico de Retro-propagación es aplicado a problemas prácticos donde se involucran redes multicapa, el entrenamiento de la red neuronal puede llegar a ser demasiado tardado. Esto se debe principalmente a que el algoritmo de Retro-propagación es una generalización del algoritmo LMS (Media de Mínimos Cuadrados), el cual garantiza la convergencia a una solución que minimice el error cuadrático medio (MSE) cuando es utilizado en redes lineales de una sola capa, Esto se logra debido a que la superficie de la función de desempeño MSE, es una función cuadrática que tiene únicamente un solo punto estacionario y la curvatura de la función en una dirección dada no presenta cambios. Pero cuando el algoritmo de Retro-propagación se usa en redes no lineales multicapas, las características de la superficie de desempeño son diferentes, ya que pueden presentar varios puntos donde se tenga un mínimo local y la curvatura puede variar ampliamente en diferentes regiones, haciendo que la convergencia a una solución que minimice al índice de desempeño MSE sea difícil de lograr. Esto ha ocasionado el surgimiento de investigaciones para acelerar la convergencia de dicho algoritmo. Estas investigaciones se han enfocado en la creación de nuevas técnicas de optimización numérica, como es el algoritmo de Levenberg-Marquardt, que es una variante del algoritmo de Retro-propagación y nos permitirá, como se ha mencionado anteriormente, minimizar funciones cuadráticas en redes multicapas que presenten funciones no lineales. Este algoritmo es el más rápido y uno de los más utilizados para entrenar redes neuronales multicapa que presentan un tamaño moderado. Su principal inconveniente radica en los requerimientos de memoria, ya que si la red presenta más de unos cientos de parámetros, el algoritmo se vuelve impráctico.

1.4.1 Matriz Jacobiana.

La matriz Jacobiana es una matriz formada por las derivadas parciales de primer orden de una función. Una de las aplicaciones más interesante de esta matriz es la posibilidad de aproximar linealmente a la función en un punto.

El paso clave en el algoritmo de Levenberg-Marquardt es la forma en cómo se desarrolla el cálculo de la matriz Jacobiana; para llevar a cabo su cálculo se tienen que realizar las derivadas de los errores con respecto a los pesos y polarizaciones de la red, por lo cual se va a conformar el siguiente vector error:

$$\mathrm{v}^T = [v_1 v_2 \ldots . v_N] = \left[e_{1,1} e_{2,1} \ldots\ e_{S^M,1}\, e_{1,2} \ldots\ e_{S^M,Q}\right] \tag{1.5}$$

Donde $N = Q$ x S^M

$Q =$ Salidas objetivo utilizadas para el entrenamiento.

$S^M =$ Neuronas contenidas en la última capa de la red neuronal.

Entonces, la matriz Jacobiana quedará expresada de la siguiente manera:

$$\mathbf{J}(\mathbf{x}) = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial w^1_{1,1}} & \frac{\partial e_{1,1}}{\partial w^1_{1,2}} \cdots \frac{\partial e_{1,1}}{\partial w^1_{S^1,R}} & \frac{\partial e_{1,1}}{\partial b^1_1} \cdots \\ \frac{\partial e_{2,1}}{\partial w^1_{1,1}} & \frac{\partial e_{2,1}}{\partial w^1_{1,2}} \cdots \frac{\partial e_{2,1}}{\partial w^1_{S^1,R}} & \frac{\partial e_{2,1}}{\partial b^1_1} \cdots \\ \vdots & \vdots \quad \vdots & \vdots \\ \frac{\partial e_{S^M,1}}{\partial w^1_{1,1}} & \frac{\partial e_{S^M,1}}{\partial w^1_{1,1}} \cdots \frac{\partial e_{S^M,1}}{\partial w^1_{S^1,R}} & \frac{\partial e_{S^M,1}}{\partial b^1_1} \cdots \\ \frac{\partial e_{1,2}}{\partial w^1_{1,1}} & \frac{\partial e_{1,2}}{\partial w^1_{1,2}} \cdots \frac{\partial e_{1,2}}{\partial w^1_{S^1,R}} & \frac{\partial e_{1,2}}{\partial b^1_1} \cdots \\ \vdots & \vdots \quad \vdots & \vdots \end{bmatrix} \tag{1.6}$$

Para calcular los elementos de la matriz Jacobiana tendremos las siguientes expresiones (1.7) y (1.8):

$$[\mathbf{J}]_{h,l} = \frac{\partial e_{k,q}}{\partial w^m_{i,j}} = \tilde{S}^m_{i,h} x a^{m-1}_{j,q} \tag{1.7}$$

Para la polarización:

$$[\mathbf{J}]_{h,l} = \frac{\partial e_{k,q}}{\partial b_i^m} = S_{i,h}^{\sim m} \tag{1.8}$$

1.4.2 Sensibilidad Marquardt.

Este es el proceso que nos da el término de Retro-propagación, porque éste describe una relación de recurrencia en la cual la sensibilidad de la última capa *M* es calculada para la sensibilidad de la capa *m+1*.

$$S_{i,h}^{\sim m} = \frac{\partial e_{k,q}}{\partial n_{i,q}^m} \tag{1.9}$$

Donde $h = (q-1)S^M + k$

Como se utiliza el término de Retro-propagación, se debe de propagar el error de la red desde la última capa hacia las demás, por tal motivo se expresa la sensibilidad de Marquardt de la última capa con la siguiente expresión (1.10), la cual será nuestro punto de inicio:

$$\mathbf{S}_q^{\sim M} = -\mathbf{F}^M\left(\mathbf{n}_q^M\right) \tag{1.10}$$

Donde $\mathbf{F}^M\left(\mathbf{n}_q^M\right)$ es la derivada de la función de activación utilizada en la última capa *M*.

Una vez obtenida la sensibilidad de Marquardt para la última capa, ésta se propagará hacia las capas ocultas de la red con la siguiente expresión:

$$\mathbf{S}_q^{\sim m} = \mathbf{F}^m\left(\mathbf{n}_q^m\right)(\mathbf{W}^{m+1})\mathbf{S}_q^{T\sim m+1} \tag{1.11}$$

1.4.3 Matriz de sensibilidad Marquardt total.

Esta matriz se forma aumentando las matrices obtenidas para cada entrada.

$$\mathbf{S}^{\sim m} = \left[\mathbf{S}_1^{\sim m}|\mathbf{S}_2^{\sim m}|\ldots|\mathbf{S}_Q^{\sim m}\right] \tag{1.12}$$

En el Apéndice A, Sección (1) se muestra un ejemplo de la obtención de la matriz Jacobiana.

1.4.4 Pasos para realizar la implementación del algoritmo de Levenberg-Marquardt.

A continuación se muestran los pasos utilizados para implementar el algoritmo de Levenberg-Marquardt.

1- Primeramente se deben de presentar todos los valores de entrada a la red neuronal, para de esta manera ir calculando el vector de salida en cada capa utilizando la expresión (1.13).

$$\mathbf{a}^{m+1} = \mathbf{f}^{m+1}(\mathbf{W}^{m+1}\mathbf{a}^{m} + \mathbf{b}^{m+1}) \tag{1.13}$$

Para m = 0,2,…, M-1, donde M es el número total de capas contenidas en la red neuronal.

Las neuronas contenidas en la primera capa recibirán los valores de entrada contenidas en el vector **p**, por lo que la expresión (1.14), es el punto de inicio de la ecuación (1.13).

$$\mathbf{a}^{0} = \mathbf{p} \tag{1.14}$$

Además, se tienen que calcular los errores obtenidos para cada valor de entrada que fue presentado a la red, utilizando la siguiente expresión:

$$\mathbf{e}_q = \mathbf{t}_q - \mathbf{a}_q^M \tag{1.15}$$

Esto con el fin de poder calcular la suma de los errores cuadráticos para todas las entradas utilizando la expresión (1.16).

$$F(\mathbf{x}) = \sum_{q=1}^{Q} (\mathbf{t}_q - \mathbf{a}_q)^T (\mathbf{t}_q - \mathbf{a}_q) \tag{1.16}$$

2- Crear la matriz Jacobiana, ecuación (1.6), e iniciar la propagación de la sensibilidad de Marquardt a partir de la última capa utilizando la ecuación (1.10) y de esta manera ir calculando las siguientes sensibilidades para las demás capas ocultas utilizando (1.11). Después, se debe de crear la matriz total de Marquardt (1.11), para finalmente de esta manera poder calcular los elementos de la matriz Jacobiana con las ecuaciones (1.7) y (1.8).

3- Resolver la ecuación para obtener $\Delta \mathbf{x}_k$.

$$\Delta \mathbf{x}_k = -[\mathbf{J}^T(\mathbf{x}_k)\mathrm{J}(\mathbf{x}_k) + \mu_k \mathbf{I}]^{-1}\mathbf{J}^T(\mathbf{x}_k)\mathbf{v}(\mathbf{x}_k)] \tag{1.17}$$

4- Recalcular la suma de los errores cuadráticos usando $\mathbf{x}_k + \Delta \mathbf{x}_k$. Si esta nueva suma es más pequeña que la calculada en el paso 1, entonces se tiene que dividir μ entre ϑ para actualizar $\mathbf{x}_{k+1} = \mathbf{x}_k + \Delta \mathbf{x}_k$ y regresar al paso 1, pero si la suma es más grande y no se reduce, entonces se tiene que multiplicar μ por ϑ y se regresa al paso 3.

El algoritmo asegura la convergencia hacia un mínimo cuando la suma de los errores cuadráticos se reduce hasta alcanzar la salida objetivo. Lo ideal es cuando la diferencia entre la salida objetivo y la salida de la red es nula.

1.5 Descripción general del FPGA (*Field Programmable Gate Array*).

Un FPGA es un circuito integrado que presenta un arreglo de bloques lógicos programables cuya interconexión y funcionalidad se puede configurar a partir de un lenguaje de descripción especializado. Fue inventado en el año 1984 por Ross Freeman y Bernard Vonderschmitt y surge de la necesidad de realizar aplicaciones similares a las de un ASIC (Application Specific Integrated Circuit), pero con la característica de ofrecer un menor costo, además de que un FPGA puede ser reprogramado ofreciendo de esta manera una mayor flexibilidad al flujo de diseño y un tiempo de desarrollo mucho menor. Otra característica de los FPGA es que se puede contar con funciones de alto nivel (sumador, multiplicador) embebidas. Los dispositivos FPGA se basan en lo que se conoce como arreglos de compuertas, la arquitectura contiene tres elementos configurables [5], como se puede observar en la Fig. (1.5).

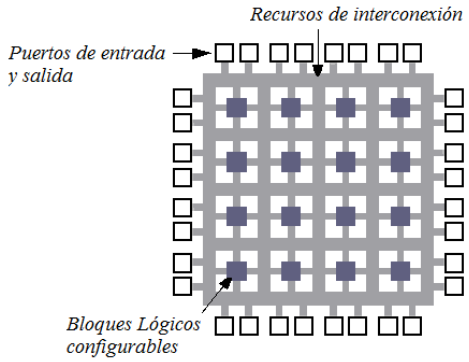


Fig. (1.5) *Elementos que conforman al FPGA.*

Los elementos que conforman la estructura principal de un FPGA son tres:

Bloques lógicos configurables (CLB): Están configurados para procesar cualquier aplicación lógica.

Puertos de entrada y salida (IOB): Permiten la conexión del exterior con los bloques programables mediante canales de conexión vertical y horizontal.

Recursos de interconexión: Permiten interconectar varios bloques lógicos entre sí, permitiendo de esta manera la elaboración de funciones complejas.

1.5.1 Aplicaciones.

Algunas aplicaciones donde se pueden utilizar los FPGA son:

Sistemas aeroespaciales y de defensa.

Reconocimiento de voz.

Sistemas de imágenes para medicina.

Sistemas de visión para computadoras.

Aplicaciones donde se requiere un alto grado de paralelismo como es la implementación de redes neuronales artificiales.

Conclusiones.

Como se pudo observar, el algoritmo de aprendizaje de Levenberg-Marquardt presenta varias ventajas ya que es un algoritmo que permite realizar el entrenamiento de una red neuronal multicapa no lineal, con un número considerable de neuronas y de forma eficiente en tiempo, como se verá en el siguiente Capítulo 3, donde se implementará este algoritmo para realizar el entrenamiento de una red neuronal multicapa utilizada como un aproximador de funciones.

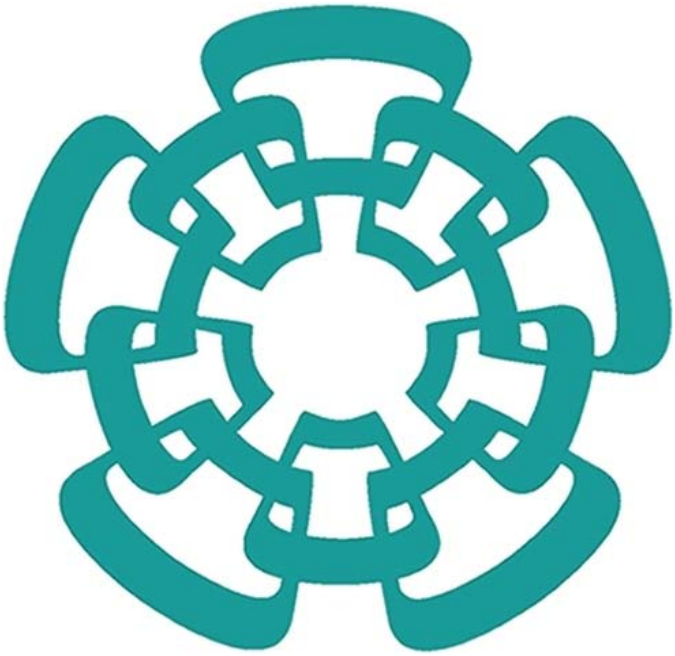
Se revisaron algunos conceptos del dispositivo digital FPGA, donde pudimos observar que este tipo de dispositivo nos ofrece la capacidad de poder realizar implementaciones de redes neuronales, con un alto grado de paralelismo lo cual, como se vió, es el punto clave de las redes neuronales.

Referencias.

1. M. Hagan, H. Demuth y M. Beale, “Neural Network Design” Oklahoma State University.
2. L. Fausett, “Fundamentals of Neural Networks Architectures, Algorithms and Applications”
3. R. Rojas, “Neural Networks A Systematic Introduction” Berlin, 1996.
4. A. Serrano, E. Soria y J. Martín “Redes Neuronales Artificiales” Departament d’ Enginyeria Electronica, Curso 2009-2010.
5. D. Maxinez y J. Alcalá “VHDL El arte de programar sistemas digitales” Instituto Tecnológico y de Estudios Superiores de Monterrey Campus México, Primera Edición 2002, Editorial Continental.

CAPÍTULO 2

Modelo análogo eléctrico de un panel solar



2.1 Introducción.

Hoy en día existen miles de aparatos, que en forma de corriente continua o de corriente alterna, utilizan la electricidad como fuente de energía. Por tal motivo su uso ha provocado un gran aumento de la demanda de consumo eléctrico. Este hecho ha propiciado la búsqueda de nuevas fuentes de energía y de nuevos sistemas de producción eléctrica, basándose fundamentalmente, en el uso de energías renovables, ya que la quema de combustibles fósiles usada actualmente, está afectando en gran manera al medio ambiente. Por este motivo, existe un gran interés en el uso de la energía solar, ya que es una fuente de energía limpia y muy amigable con el medio ambiente. Este tipo de fuente de energía se puede aprovechar a partir del uso de *métodos indirectos*, en los cuales la energía solar es utilizada para calentar un fluido (agua) y convertirlo en vapor, con el fin de que la generación de la electricidad se produzca con el movimiento de un alternador, o a partir de *métodos directos*, en los cuales la luz del Sol es convertida directamente en electricidad, mediante el uso de celdas solares.

Actualmente existen muchas aplicaciones enfocadas al uso de la energía solar como son:

- Luces de señalización.
- Telecomunicaciones remotas.
- Bombas para agua potable o irrigación.
- Generación de energía a gran escala para la red eléctrica.
- Electrificación de zonas rurales y aisladas.

Además de que los sistemas fotovoltaicos presentan grandes beneficios en comparación con otras fuentes de energía, algunas de las ventajas son:

- No requieren combustibles.
- Mínimo mantenimiento.
- Fuente inagotable de energía.
- Larga vida de operación.
- No contaminan.
- Sistemas silenciosos.
- Equipo resistente al medio ambiente.

En este capítulo se tratará el procedimiento utilizado, para el desarrollo en lenguaje Spice, de un modelo análogo eléctrico, el cual nos permitirá emular el comportamiento del panel solar ST5 de la compañía Siemens para condiciones arbitrarias de irradiancia (G) y temperatura (T), para así de esta manera, obtener los parámetros eléctricos: corriente a corto circuito (I_{SC}) y voltaje a circuito abierto (V_{OC}) de dicho panel solar. Estos dos parámetros, en combinación con la irradiancia y la temperatura, nos servirán para realizar el entrenamiento de una red neuronal artificial, como se verá en el Capítulo 3.

2.2 Celda Solar.

Un sistema fotovoltaico está conformado por un conjunto de dispositivos (regulador, inversor, batería, panel solar), cuya función es la de transformar en forma directa la energía solar en energía eléctrica. El componente clave de todo sistema fotovoltaico y responsable de esta conversión, es el panel o módulo solar, el cual a su vez está formado por la asociación de varias células o celdas solares conectadas en formal serial o en forma paralela, dependiendo de la aplicación [2]. Las celdas solares son las encargadas de convertir los fotones de la luz solar en electricidad, basándose en el principio del efecto fotovoltaico [3], el cual se basa en la propiedad que tienen los materiales semiconductores de aumentar la densidad de electrones libres bajo ciertos estímulos externos como son: cuando se eleva su temperatura o cuando se les ilumina. Este efecto fotovoltaico establece que la emisión de electrones es producto de la absorción de fotones de luz, ya que las celdas solares se basan en uniones de tipo p-n, desplazando a los electrones de la banda de valencia a la banda de conducción, por el aporte energético de los fotones incidentes.

2.2.1 Celda Solar Ideal.

El comportamiento de una celda solar puede ser representado con un modelo de primer orden, en el cual, se describe la respuesta de un dispositivo solar a dos tipos de excitaciones: voltaje y luz, donde una corriente de iluminación o fotocorriente (I_L), debida a la incidencia de fotones, circula desde la región n a la región p de un material semiconductor y otra corriente denominada de obscuridad o corriente de diodo (I_D), originada por la recombinación de portadores favorecida por la tensión en el circuito externo, circula desde la región p hacia la región n. El valor de la corriente total generada por la celda solar es expresado con la ecuación (2.1), en la cual se puede

observar que la corriente de salida de la celda solar I, va a ser igual a la diferencia entre la corriente fotogenerada I_L y la corriente del diodo I_D.

$$I = I_L - I_D \tag{2.1}$$

Donde:

$$I_D = I_0\left(e^{\frac{V}{V_T}} - 1\right) \tag{2.2}$$

Al tratarse de una celda solar ideal, sin pérdidas resistivas, la corriente fotogenerada I_L será equivalente a la corriente a corto circuito I_{SC}, por lo cual sustituyendo (2.2) en (2.1) tendremos:

$$I = I_{SC} - I_0\left(e^{\frac{V}{V_T}} - 1\right) \tag{2.3}$$

Donde:

I - Corriente generada por la celda solar.

I_{SC} - Corriente a corto circuito.

I_0 - Corriente inversa de saturación del diodo.

V - Voltaje aplicado.

V_T - Voltaje térmico.

Este modelo es el más sencillo y el más utilizado cuando se trabaja con celdas solares, debido a que no toma en cuenta las pérdidas resistivas que presenta una celda real. Además, este modelo debido a su simplicidad puede ser fácilmente representado eléctricamente utilizando lenguaje Spice, por una fuente de corriente de valor I_{SC} en paralelo con un diodo, ver Fig. (2.1).

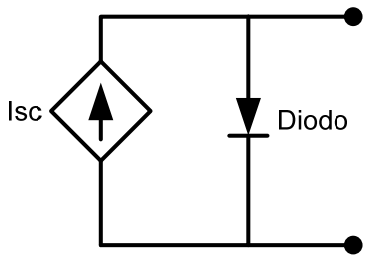


Fig. (2.1) *Representación eléctrica de una celda solar.*

2.2.2 Parámetros eléctricos de una celda solar.

Existen ciertos parámetros eléctricos que se pueden obtener a la salida de una celda solar, los cuales serán de utilidad a lo largo de este capítulo.

Corriente a corto circuito (I_{SC}): Es el máximo valor de corriente que puede circular por la celda solar; se da cuando sus terminales están cortocircuitadas, Fig. (2.2), ocurre cuando $V = 0$ en la ecuación (2.3).

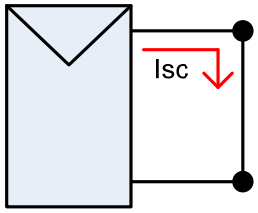


Fig. (2.2) *Símbolo eléctrico de una celda solar en configuración de corto circuito.*

Voltaje a circuito abierto (V_{CO}): Es la máxima tensión que se obtiene en los extremos de la celda solar, que se da cuando no está conectada a ninguna carga Fig. (2.3), ocurre cuando se tiene la condición $I = 0$ en la ecuación (2.3).

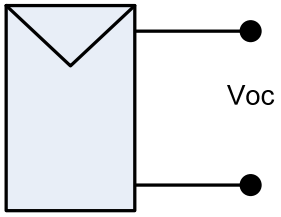


Fig. (2.3) *Símbolo eléctrico de una celda solar en configuración de circuito abierto.*

Corriente de máxima potencia (I_{mm}): Es el valor nominal de la corriente, en el punto donde se obtiene la máxima potencia.

Voltaje de máxima potencia (V_{mm}): Es el valor nominal de la tensión, en el punto donde se obtiene la máxima potencia.

2.2.3 Curva característica I - V.

La representación gráfica que permite observar los parámetros eléctricos descritos anteriormente, se denomina *curva característica* o curva *corriente – tensión* [4] la cual se presenta en la Fig. (2.4).

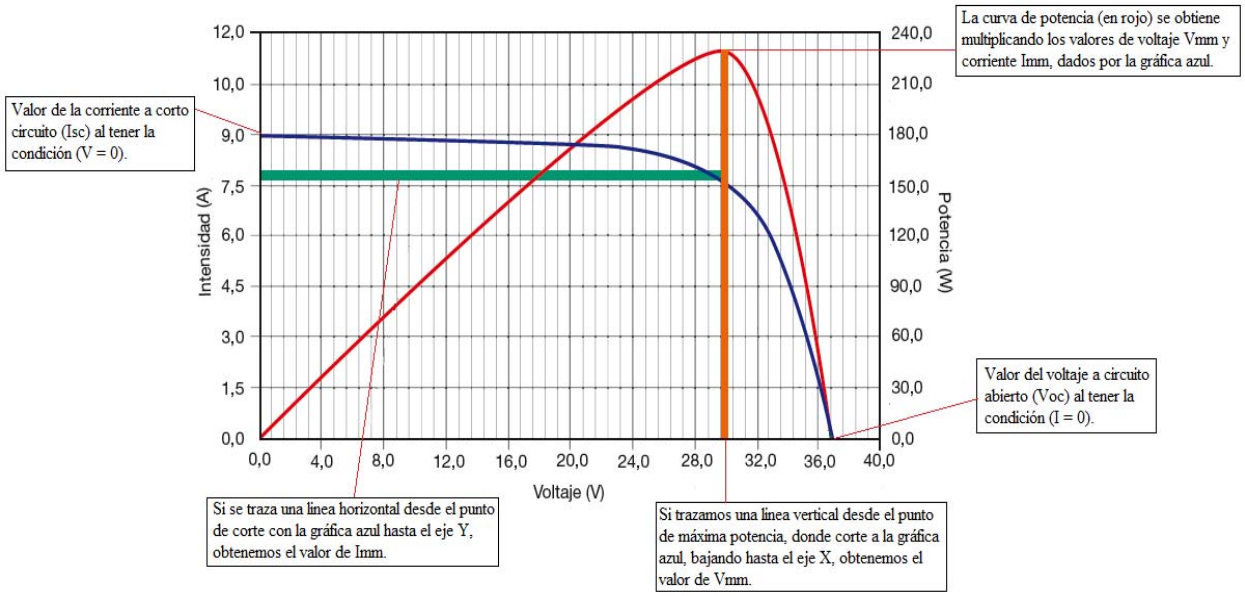


Fig. (2.4) *Curva característica.*

2.2.4 Configuración: (Serie /Paralelo).

La potencia que proporciona una celda solar de tamaño estándar es muy pequeña (en torno a 1 o 2W), por lo que generalmente será necesario tener que asociar varias celdas, con el fin de proporcionar la potencia necesaria para distintas aplicaciones. Se pueden encontrar diferentes posibilidades:

Conexión en serie: este tipo de conexión permite aumentar el valor de la tensión final, ya que en la mayoría de las aplicaciones fotovoltaicas, se requieren tensiones mayores a la decena de voltios.

Conexión en paralelo: este tipo de conexión permite aumentar el valor de la corriente final.

Conexión mixta: este tipo de conexión permite incrementar tanto el valor de la tensión, como el valor de la corriente; esto se logra teniendo paneles solares conformados por cadenas de celdas

solares conectadas en paralelo, donde cada cadena va a estar conformada por celdas solares conectadas en forma serial.

2.3 Modelo análogo eléctrico de un panel solar.

Al no contar físicamente con un panel solar para la obtención de los parámetros eléctricos: *corriente a corto circuito* (I_{SC}) y *voltaje a circuito abierto* (V_{OC}), se planteó el uso de un *modelo análogo eléctrico* o *de comportamiento*, el cual permitirá emular la respuesta de cualquier panel fotovoltaico bajo condiciones arbitrarias de irradiancia (G) y de temperatura (T). La obtención de los valores de la corriente a corto circuito y del voltaje a circuito abierto, va a ser de ayuda para realizar el entrenamiento de una red neuronal artificial como se explicó anteriormente. Una de las ventajas que presenta este modelo, es que puede ser utilizado para emular el comportamiento de paneles solares, que presenten celdas solares de distintos materiales a los utilizados comúnmente: Silicio (Si), Germanio (Ge) y Arseniuro de Galio (AsGa); esto es importante porque permite omitir ciertos parámetros físicos, que se requieren para reproducir el comportamiento de una celda solar, ya que en algunas ocasiones los valores de estos parámetros no se encuentran disponibles para los usuarios [1]. Por tal motivo, este modelo permitirá utilizar únicamente los parámetros eléctricos más usuales, que se encuentran disponibles en una hoja de datos de un panel solar como son: corriente a corto circuito, voltaje a circuito abierto, potencia máxima y los coeficientes de temperatura tanto de la corriente a corto circuito, como del voltaje a circuito abierto.

2.3.1 Panel Solar ST5.

Para nuestro propósito se utilizó el modelo eléctrico desarrollado, para emular la respuesta del panel solar ST5 de la compañía Siemens, el cual está compuesto por una estructura monolítica de 33 celdas, las cuales se encuentran conectadas en forma serial y están elaboradas de Cobre, Indio y Diselenio (CIS). Otras características que presenta este panel solar se pueden observar en la hoja de datos en el Apéndice B, Sección 1.

2.3.2 Base de datos Meteorológica.

Para realizar la simulación del modelo eléctrico, primeramente se consiguió una base de datos la cual está conformada por los parámetros climáticos: irradiancia y temperatura, ya que como se observó anteriormente, estos parámetros, son de suma importancia para el funcionamiento de un panel solar y para la obtención de sus parámetros eléctricos de salida. La base de datos con la que se trabajó es una base de datos que se obtuvo del Sistema Estatal de Monitoreo Agro-Climático del Estado de Nayarit, en conjunto con el Instituto Nacional de Investigaciones Forestales, Agrícolas y Pecuarias (Inifap) y la SAGARPA. Esta base de datos tiene una fecha de inicio del 1° de Enero del 2010 y una fecha de término del 1° de Enero del 2013, lo que nos da un total de 1097 días de monitoreo (3 años). Cabe mencionar que la base de datos que se manejó muestra la irradiancia y la temperatura media diaria.

2.3.3 Unidades de las variables.

Debido a que el simulador eléctrico (ELDO) de Mentor Graphics, utiliza únicamente unidades eléctricas para trabajar y como se desea saber la respuesta del modelo eléctrico ante los valores de entrada: irradiancia y temperatura, se tuvo que realizar la normalización de las unidades por lo que se realizó la siguiente conversión de unidades:

Para la irradiancia: $1\ {}^{W}/_{m^2} = 1\ V$

Para la temperatura: $1\ °C = 1\ V$

2.3.4 Unidades de tiempo.

Ya que el objetivo del uso del simulador eléctrico es el de analizar la respuesta de un panel solar realizando un análisis transitorio en el tiempo, se debe de realizar un cambio en la unidad de tiempo. Debido a que existen dos tipos de unidades de tiempo: la unidad de tiempo interno del simulador ELDO, la cual se maneja en segundos y la unidad de tiempo del panel solar, la cual se encuentra en días, se propone la equivalencia donde: 1 día = 1 μs. Esto permite que el tiempo de procesamiento utilizado para la simulación del CPU se reduzca considerablemente, permitiendo hacer este tipo de simulaciones en cualquier equipo de cómputo.

Podemos observar en la Fig. (2.5) y en la Fig. (2.6) los perfiles de irradiancia y temperatura, que se obtuvieron de la base de datos meteorológica, los cuales van a servir como entradas para el modelo eléctrico propuesto, como se observará más adelante.

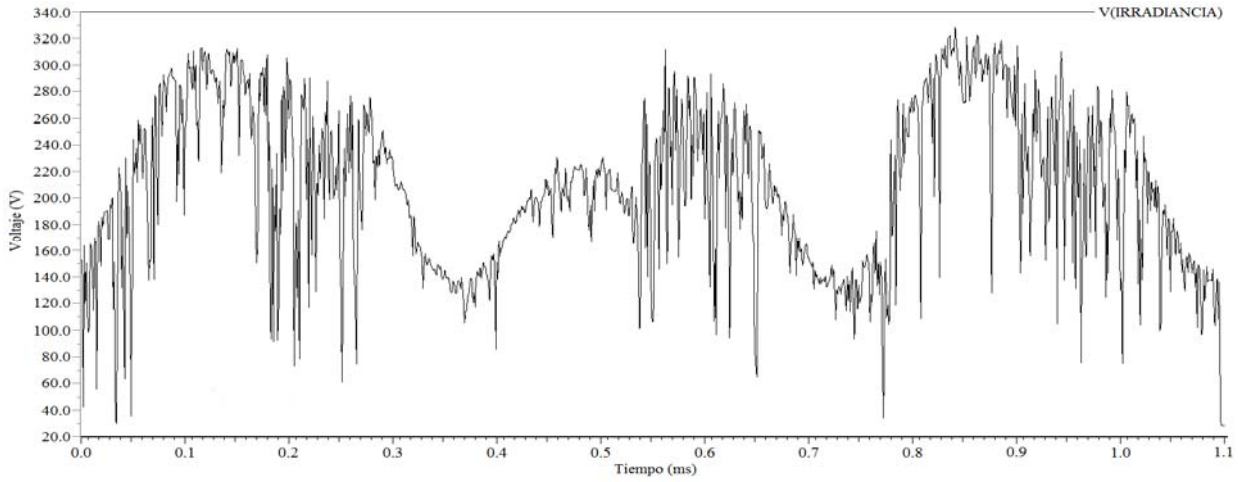


Fig. (2.5) *Perfil de monitoreo de la irradiancia.*

Los valores de irradiancia del Estado de Nayarit, presentan el siguiente rango de operación $28.1\ W/m^2 \leq G \leq 329\ W/m^2$.

Para el perfil de la temperatura, Fig. (2.6), se obtuvo un rango de operación que oscila entre $11.8\ °C\ \leq T \leq 39.6°C$.

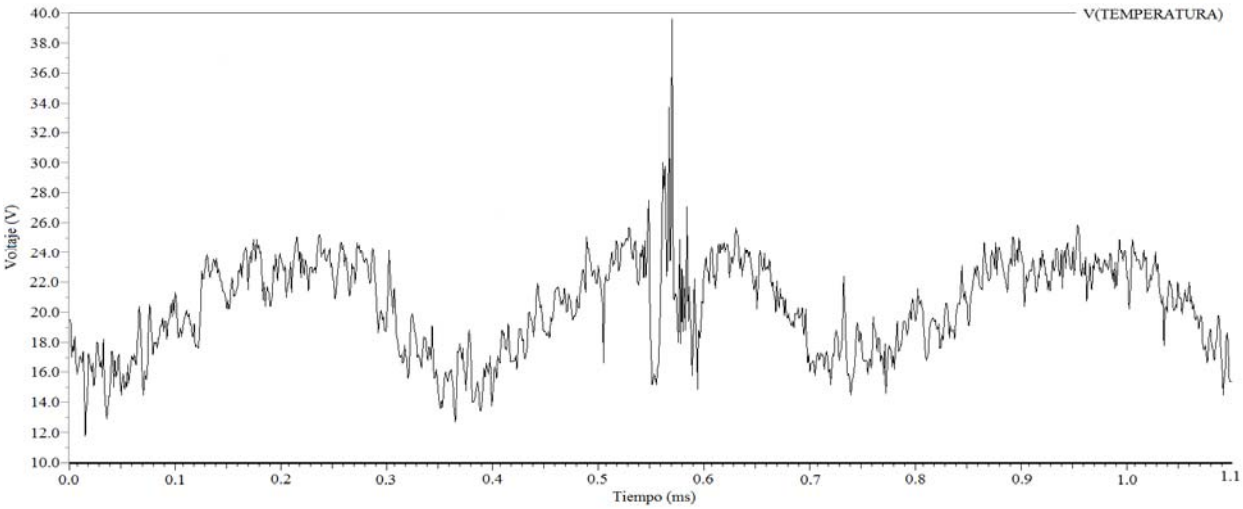


Fig. (2.6) *Perfil de monitoreo de la temperatura.*

Para realizar la implementación del modelo eléctrico se realizó el código denominado `MODULE_BEH.LIB`, usando código Spice, el cual se puede observar en el Apéndice B, Sección2.

2.4. Subcircuito Spice.

Utilizando el programa de desarrollo Mentor Graphics se realizó el siguiente subcircuito, Fig.(2.7), donde se pueden observar los parámetros utilizados como entradas y los parámetros utilizados como salidas del modelo eléctrico empleado.

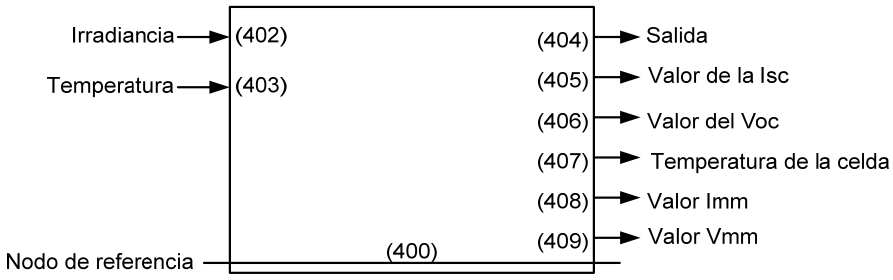


Fig. (2.7) *Subcircuito del modelo eléctrico de un panel solar.*

Dentro de este subcircuito se describió el siguiente diagrama eléctrico, Fig. (2.8), el cual está conformado por resistencias y dispositivos de tipo G (fuentes de corriente controladas por voltaje) y dispositivos de tipo E (fuentes de voltaje controladas por voltaje), las cuales son fuentes de tipo PWL (lineal por tramos), que se encuentran disponibles en la biblioteca de Spice. Estas fuentes permiten el manejo de formas arbitrarias, como es el caso de los valores de entrada, además de que se pueden utilizar para la obtención de los parámetros de salida del subcircuito.

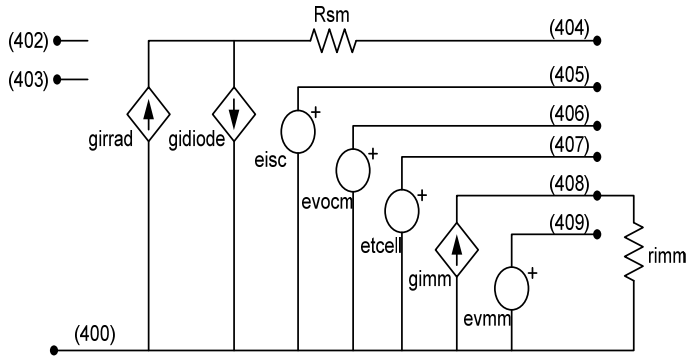


Fig. (2.8) *Diagrama esquemático del modelo eléctrico de un panel solar.*

Este modelo asume que se puede emular el comportamiento de un panel solar utilizando únicamente dos fuentes de corriente controladas por voltaje (girrad y gidiode), conectadas en serie con una resistencia R_{sm}. Con esta configuración se trata de reproducir el comportamiento de un panel solar compuesto por celdas solares ideales, las cuales como se vio anteriormente, únicamente se componen por una fuente de corriente, la cual imita la corriente de iluminación y por un diodo que imita la corriente de obscuridad, que en nuestro caso, al no tratarse de un panel solar conformado de celdas de Silicio, Germanio o Arseniuro de Galio, el modelo del diodo que se maneja en la biblioteca de Spice no es de gran utilidad, por lo que en la literatura [1], se maneja la respuesta del diodo, utilizando una fuente de corriente denominada (gidiode), la cual es descrita con la ecuación (2.4).

$$i(gidiode) = \frac{I_{SC}}{\left(e^{\frac{V_{OC}}{V_T}} - 1\right)}\left(e^{\frac{V}{V_T}} - 1\right) \tag{2.4}$$

2.4.1 Corriente a corto circuito.

Para calcular el valor de la corriente a corto circuito ante cualquier valor de irradiancia, se puede usar la expresión (2.5), la cual en el listado mencionado anteriormente, se declaró dentro de una fuente de corriente denominada (eisc); esta misma expresión es usada en la fuente de corriente denominada (girrad):

$$I_{SC} = \frac{I_{SCMr}}{1000} G + Coef\_I_{SC}(T_{CELL} - T_r) \tag{2.5}$$

Donde:

I_{SCMr} – Corriente a corto circuito obtenido de la base de datos.

G - Valor de la irradiancia del nodo 402.

$Coef\_I_{SC}$ – Valor del coeficiente de temperatura de la corriente a corto circuito.

T_{CELL} – Temperatura de la celda.

T_r – Temperatura de referencia (25°C).

El perfil de la corriente a corto circuito, que se obtuvo del análisis transitorio se muestra en la Fig. (2.9).

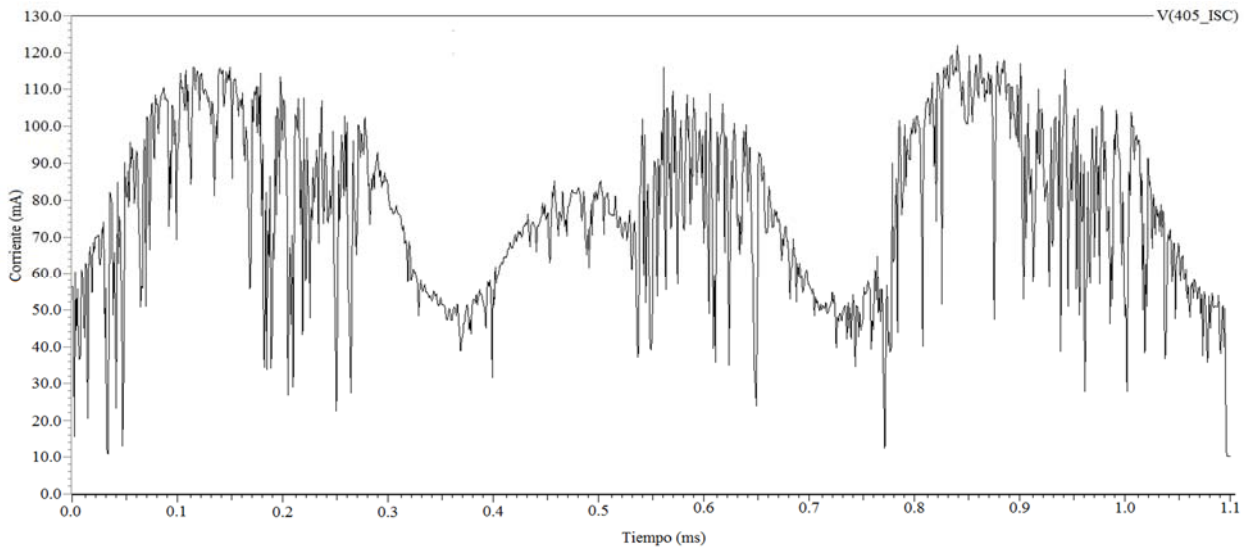


Fig. (2.9) *Corriente a corto circuito I_{SC} del nodo (405).*

2.4.2 Temperatura de la celda solar.

La temperatura de la celda T_{CELL} se obtiene utilizando la expresión (2.6), la cual se describió en la fuente de voltaje denominada (etcell), donde el parámetro T_a representa la temperatura que se obtuvo de la base de datos, correspondiente al nodo (403), Fig. (2.10).

$$T_{CELL} - T_a = \frac{NOCT - 20}{800} G \tag{2.6}$$

El término $NOCT$ (Temperatura Nominal de Funcionamiento de la celda) se refiere a la temperatura a la que operan las celdas solares bajo ciertas condiciones de operación estándar (SOC), las cuales son:

- Irradiancia de 800 $^{W}/_{m^2}$
- Temperatura ambiente 20 °C
- Velocidad media del viento 1 $^{m}/_{s}$
- Conexión a circuito abierto de la celda o panel solar, con todas las partes expuestas al viento.

Este parámetro ayuda a relacionar la temperatura ambiente con la temperatura real de funcionamiento de la celda; el valor $NOCT$ que se utilizó y se especifica en la hoja de datos para este panel solar fue de 47°C.

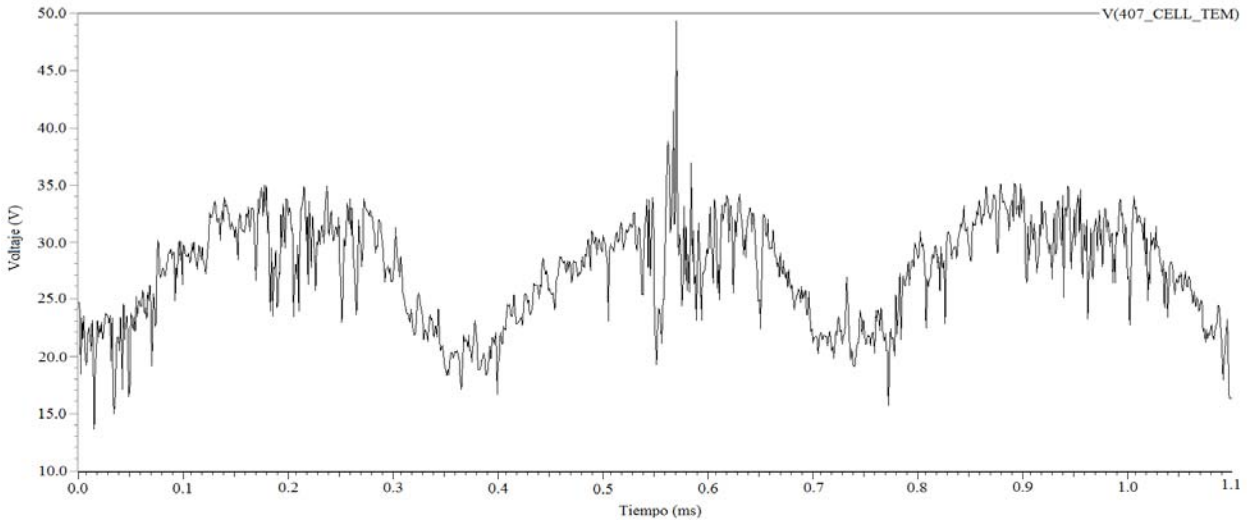


Fig. (2.10) *Temperatura de la celda solar T_{CELL} del nodo (407).*

2.4.3 Voltaje a circuito abierto.

Para el voltaje a circuito abierto del panel solar Fig. (2.11), se declaró la expresión (2.7), dentro de la fuente de voltaje denominada (evocm).

$$V_{OC} = V_{OCMr} + Coef\_V_{OC}(T_{CELL} - T_r) + V_T In \frac{I_{SC}}{I_{SCMr}} \qquad (2.7)$$

Donde:

V_{OCMr} - Voltaje a circuito abierto del panel solar obtenido de la hoja de datos.

$Coef\_V_{OC}$ – Valor del coeficiente de temperatura del voltaje a circuito abierto.

V_T – Voltaje térmico.

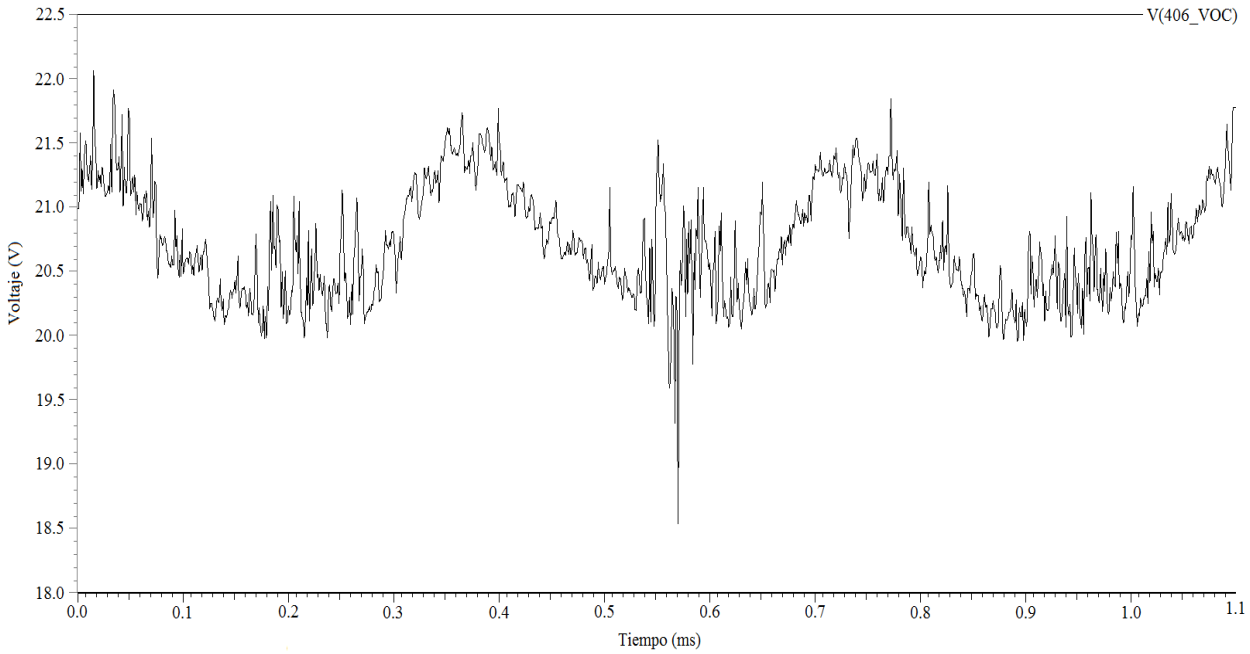


Fig. (2.11) *Voltaje a circuito abierto V_{OC} del nodo (406).*

2.4.4 Corriente - Voltaje del punto de máxima potencia.

Otro par de parámetros que son de suma importancia en la implementación de un sistema fotovoltaico son: el valor de la corriente en el punto de máxima potencia I_{mm}, el cual se representa usando una fuente de corriente controlada por voltaje denominada (gimm), expresión (2.8) y el valor del voltaje en el punto de máxima potencia V_{mm}, el cual se genera con una fuente de voltaje controlada por voltaje denominada (evmm), expresión (2.9). Estos dos parámetros son importantes, porque indican el punto o valor donde se tiene la máxima potencia de un panel solar.

$$I_{mm} = I_{mmr}\frac{G}{1000} + Coef\_I_{SC}(T_{CELL} - T_r) \tag{2.8}$$

$$V_{mm} = N_s V_T In\left(1 + \frac{I_{SC} - I_{mm}}{I_{SC}}\left(e^{\frac{V_{OC}}{N_S V_T}} - 1\right)\right) - I_{mm}R_{sm} \tag{2.9}$$

Donde I_{mmr} representa la corriente del punto de máxima potencia y este parámetro es obtenido de la hoja de datos. Debido a que nuestro panel solar, presenta 33 celdas conectadas en forma serial, este valor es representado en la expresión (2.9), con la variable N_S, además el valor de la resistencia R_{sm} se obtiene utilizando la ecuación (2.10).

$$R_{sm} = \frac{V_{OC}}{I_{SC}} - \frac{P_{max}}{FF_0 I_{SC}^2} \tag{2.10}$$

FF_0 se denomina factor de forma, el cual se define como la razón entre la potencia máxima que puede entregar la celda solar a la carga y el producto $I_{SC}V_{OC}$, el subíndice 0 hace referencia a una celda solar ideal. Esto significa que no se toman en cuenta las pérdidas resistivas que existen en una celda real; el factor de forma se puede representar con la siguiente expresión:

$$FF_0 = \frac{v_{oc} - In(v_{oc} + 0.72)}{1 + v_{oc}} \tag{2.11}$$

El parámetro v_{oc} es el valor normalizado del voltaje a circuito abierto, el cual se define con la expresión (2.12).

$$v_{oc} = \frac{V_{OC}}{V_T} \tag{2.12}$$

Además el parámetro P_{max}, es denominado como la potencia máxima del panel solar y su valor se obtiene de la hoja de datos, expresión (2.10). Ya que en este modelo se trabaja con valores de voltaje tanto en la entrada como en la salida, para obtener el valor de I_{mm}, nodo (408), se tuvo que agregar una resistencia denominada (rimm), para así utilizar la ley de Ohm.

En la Fig. (2.12) y la Fig. (2.13) se pueden apreciar los perfiles que se obtuvieron para la corriente y el voltaje en el punto de máxima potencia.

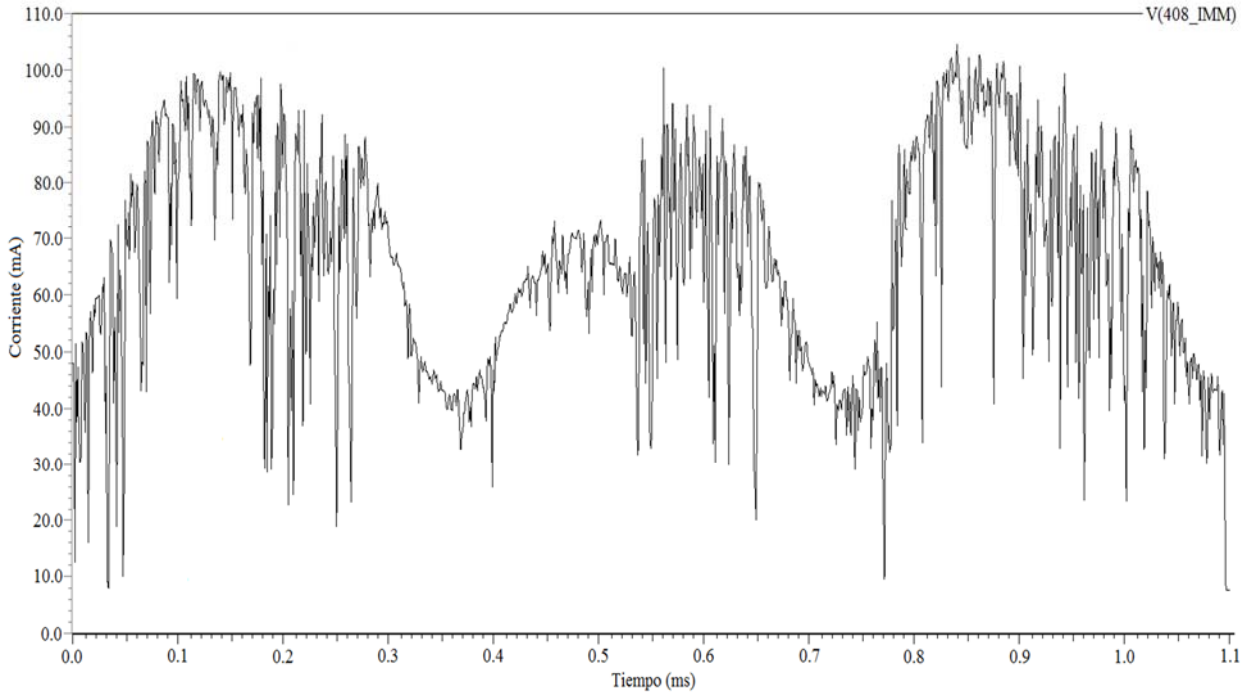


Fig. (2.12) *Corriente en el punto de máxima potencia del nodo (408).*

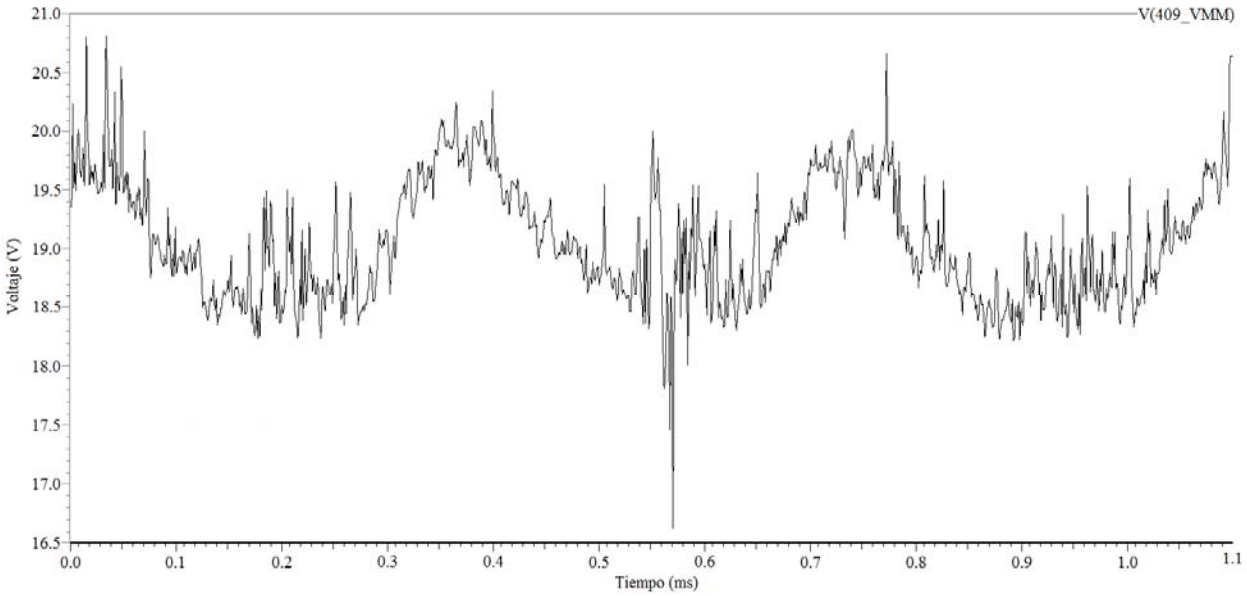


Fig. (2.13) *Voltaje en el punto de máxima potencia del nodo (409).*

Conclusiones.

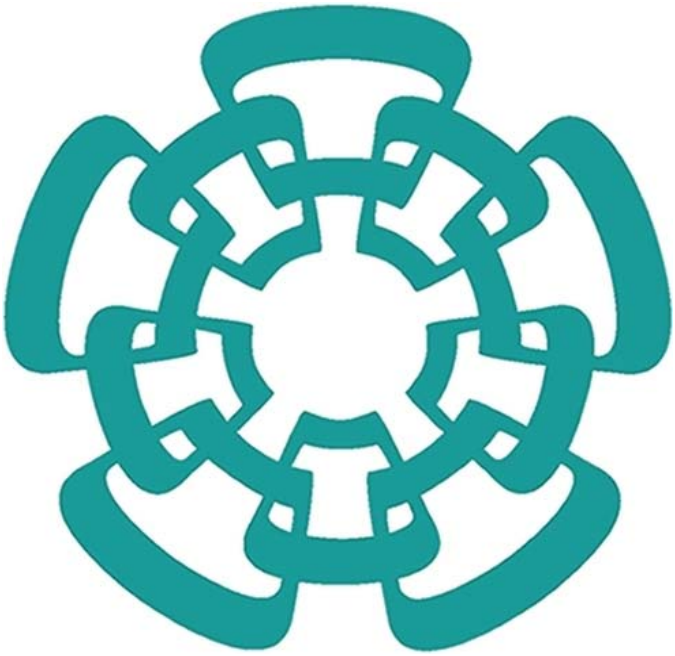
En este capítulo, se resolvió el problema de la obtención de los parámetros eléctricos de un panel solar. Esto se logró implementando en lenguaje Spice, un modelo análogo eléctrico, el cual imita la respuesta que tendría un panel solar real, ante valores arbitrarios de irradiancia y de temperatura. Para nuestro propósito se utilizaron los parámetros de la hoja de datos del panel solar ST5; se observó que este modelo permite trabajar los datos de cualquier panel sin importar su composición, además de que se basa en el comportamiento de celdas solares ideales. Se realizó un análisis transitorio, donde se obtuvieron los perfiles de la corriente a corto circuito y del voltaje a circuito abierto, comprobando de esta manera el buen funcionamiento de dicho modelo.

Referencias

1. L. Castañer and S. Silvestre, "Modelling Photovoltaic Systems using PSpice", Universidad Politecnica de Cataluña, Barcelona, Spain, Jonh Wiley & Sons.
2. O. Perpiñan Lamigueiro, "Energía Solar Fotovoltaica", Cerative Commons, Enero 2012.
3. J. L. Balenzategui Manzanares, "Fundamentos de la Conversión Fotovoltaica: La Célula Solar."
4. H. L. Gasquet, Conversión de la Luz Solar en Energía Eléctrica Manual Teórico Práctico sobre los Sistemas Fotovoltaicos", Solartronic.

CAPÍTULO 3

Implementación de una red neuronal en Matlab



3.1 Introducción.

Hoy en día existe un gran interés en el uso de las redes neuronales artificiales (RNA's), debido a que están inspiradas en las redes neuronales biológicas [1]. Entre sus funciones más comunes, como se analizó anteriormente, se encuentran, la del *aprendizaje* a partir de la experiencia, ya que al tener un conjunto de entradas las neuronas se ajustan para producir salidas consistentes, la *generalización*, ya que pueden ofrecer dentro de un margen, respuestas correctas a entradas que presentan pequeñas variaciones y finalmente, la *abstracción* ya que pueden aislar o considerar por separado las cualidades de un objeto [2].

En el campo del diseño y la investigación de las redes neuronales artificiales, un factor importante es la creación de distintas arquitecturas y el uso de algoritmos de aprendizaje, para la realización de distintas aplicaciones. Una herramienta de desarrollo que nos va a permitir profundizar en la implementación de las redes neuronales va a ser el *toolbox* de redes neuronales de Matlab versión R2012a, ya que a partir del uso de funciones específicas se va a poder implementar cualquier tipo de arquitectura neuronal que se desee, así como el desarrollo de su entrenamiento más óptimo.

En este capítulo trataremos la creación y el entrenamiento de una red neuronal artificial multicapa, de tipo feedforward, con una arquitectura (2-7-9-2), la cual servirá para aproximar los perfiles de la corriente a corto circuito y el voltaje a circuito abierto de un panel solar. Esta red neuronal será entrenada con el algoritmo de aprendizaje de Levenberg-Marquardt, aplicando un tipo de entrenamiento supervisado, para la obtención de los valores óptimos de los pesos y polarización. Además se llevará a cabo un análisis del proceso de entrenamiento con ayuda de herramientas estadísticas, para así determinar si la red neuronal se entrenó correctamente.

3.2 Recolección de datos.

Un paso importante que se debe realizar antes de hacer la implementación de una red neuronal artificial en la herramienta de desarrollo de Matlab, es la recolección de los datos, los cuales serán de gran utilidad para el entrenamiento de la red en consideración. Como se ha explicado se va a realizar un entrenamiento supervisado, donde para un patrón de entrada se va a tener un patrón de salida objetivo. Para el problema de aproximación del módulo fotovoltaico se tomarán

los datos obtenidos de la base de datos meteorológica (*irradiancia y temperatura*) como patrón de entrada y como patrón de salida o target, los datos obtenidos del modelo análogo – eléctrico del módulo solar comercial de matrícula: ST5 (*corriente y voltaje*), que se desarrolló en el capítulo anterior. Así se conformó una base de datos analógicos de un tamaño de [1097 x 4], donde los 1097 renglones hacen referencia a los 1097 días de medición, las primeras dos columnas representan datos de entrada y las últimas dos columnas datos de salida objetivo.

3.3 Pre-procesamiento y Pos-procesamiento de los datos.

El pre-procesamiento y el pos-procesamiento de los datos de entrada y salida es muy útil debido a que permite que el entrenamiento de la red sea más eficiente: Este proceso consiste en la eliminación del ruido (*valores desconocidos, inconsistentes o repetitivos*) de la base de datos, ya que esta clase de valores no proporcionan información útil a la red, además durante el pre-procesamiento se tiene que llevar a cabo la normalización de los datos en un cierto intervalo. Esto se realiza para evitar la saturación de nuestra función de transferencia tangente hiperbólica sigmoidal (*tangsig*), ya que ésta, al tener un valor de entrada neta (*n*) mayor a +/- 3.5, inmediatamente se tendrá a la salida de la neurona un valor saturado de +/- 1, Así con la normalización de los datos se pretende que los valores de salida de las neuronas, estén dentro de un intervalo y no más allá de éste [3]. Esto va a facilitar que la red neuronal generalice los datos de una mejor manera. En nuestro caso, la base de datos que se utilizó no presentó valores inconsistentes ni valores desconocidos, por lo que únicamente se normalizaron los datos de entrada y salida objetivo en un interval**o** de +/-1, utilizando la función (*mapminmax*) la cual utiliza la siguiente ecuación (3.1).

$$y = (ymax - ymin) * \frac{(x - xmin)}{(xmax - xmin)} + ymin \qquad (3.1)$$

Donde:

x = valor a normalizar.

ymax = valor máximo al que se va a normalizar en este caso +1.

ymin = valor mínimo al que se va a normalizar en este caso -1.

xmax = valor máximo de la base de datos.

xmin = valor mínimo de la base de datos.

Cuando una red neuronal ha sido entrenada, esto significa que se han obtenido los valores óptimos de los pesos y polarización, con los cuales se tiene el error mínimo entre la salida de la red neuronal y la salida objetivo, se puede realizar el proceso de pos-procesamiento de los datos, el cual consiste en realizar la desnormalización de los datos, para de esta manera obtener la magnitud original de la base de datos; en nuestro caso este proceso se realizará en el siguiente capítulo como se explicará más adelante.

3.4 División de los datos.

Al tener la base de datos conformada y pre-procesada, ésta se tuvo que dividir en tres subconjuntos denominados: *Entrenamiento, Validación* y *Prueba*, los cuales como sus nombres nos indican, sirvieron para realizar el monitoreo de los procesos de entrenamiento, validación y prueba de la red neuronal.

3.4.1 Subconjunto de Entrenamiento.

Este subconjunto es el mayor de los tres y sirve durante el proceso de entrenamiento de la red neuronal, ya que permitirá calcular el gradiente e ir actualizando en cada iteración el valor de los pesos y las polarizaciones. A este subconjunto se le asignó el 70% de la base total, es decir 767 muestras.

3.4.2 Subconjunto de Validación.

Este subconjunto es de gran importancia debido a que ayuda a monitorear el proceso de entrenamiento, ya que durante la primera fase del entrenamiento, el error de validación y el error de entrenamiento presentan una disminución en cada iteración, pero cuando existe un sobreajuste de los datos, esto quiere decir que la red empieza a memorizar puntos individuales en vez de generales. El error en la validación empieza a incrementarse, haciendo que se detenga el proceso de entrenamiento y se salven los últimos valores de los pesos y polarización que se obtuvieron antes de que se empezara a incrementar el error de validación. Para este subconjunto se asigna el 15% de la base de datos, es decir 165 muestras.

3.4.3 Subconjunto de Prueba.

Este subconjunto es presentado una vez que se ha entrenado la red neuronal, y sirve para evaluar la respuesta de la red ante valores que nunca observó durante el entrenamiento. A este subconjunto se le asignó el 15% de la base de datos, es decir 165 muestras.

Para realizar la división de los datos, existen varios comandos disponibles en Matlab como son por bloques, por intervalos, por índice y aleatoriamente, por lo que el último método fue el preferido, para esto se utilizó la función (*dividerand*), como se puede ver en el Apéndice B, Sección 1 línea 47.

3.5 Creación de la red.

Para realizar la implementación de una red neuronal, se realizó una plantilla por medio de un script en Matlab, el cual permitió a través de funciones, la creación de una red neuronal desde cero. Esto quiere decir que se puede realizar cualquier tipo de arquitectura que se desee, sin importar el número de capas, de neuronas o el tipo de interconexión entre capas. La arquitectura de red neuronal que aquí se propone para la solución del problema de aproximación de funciones, es una arquitectura multicapa, de tipo *feedforward* con la siguiente configuración (2-7-9-2). Esta arquitectura se puede observar en la Fig.(3.1) la cual está conformada por un vector de entrada de dos componentes (*irradiancia y temperatura*), un vector de salida de dos componentes (*corriente a corto circuito y voltaje a circuito abierto*). Además, esta red cuenta con 2 capas ocultas, una capa compuesta por 7 neuronas y otra con 9 neuronas. Por último, se tienen 2 neuronas en la capa de salida, obteniendo de esta manera un total de 18 neuronas, 95 pesos y 18 polarizaciones.

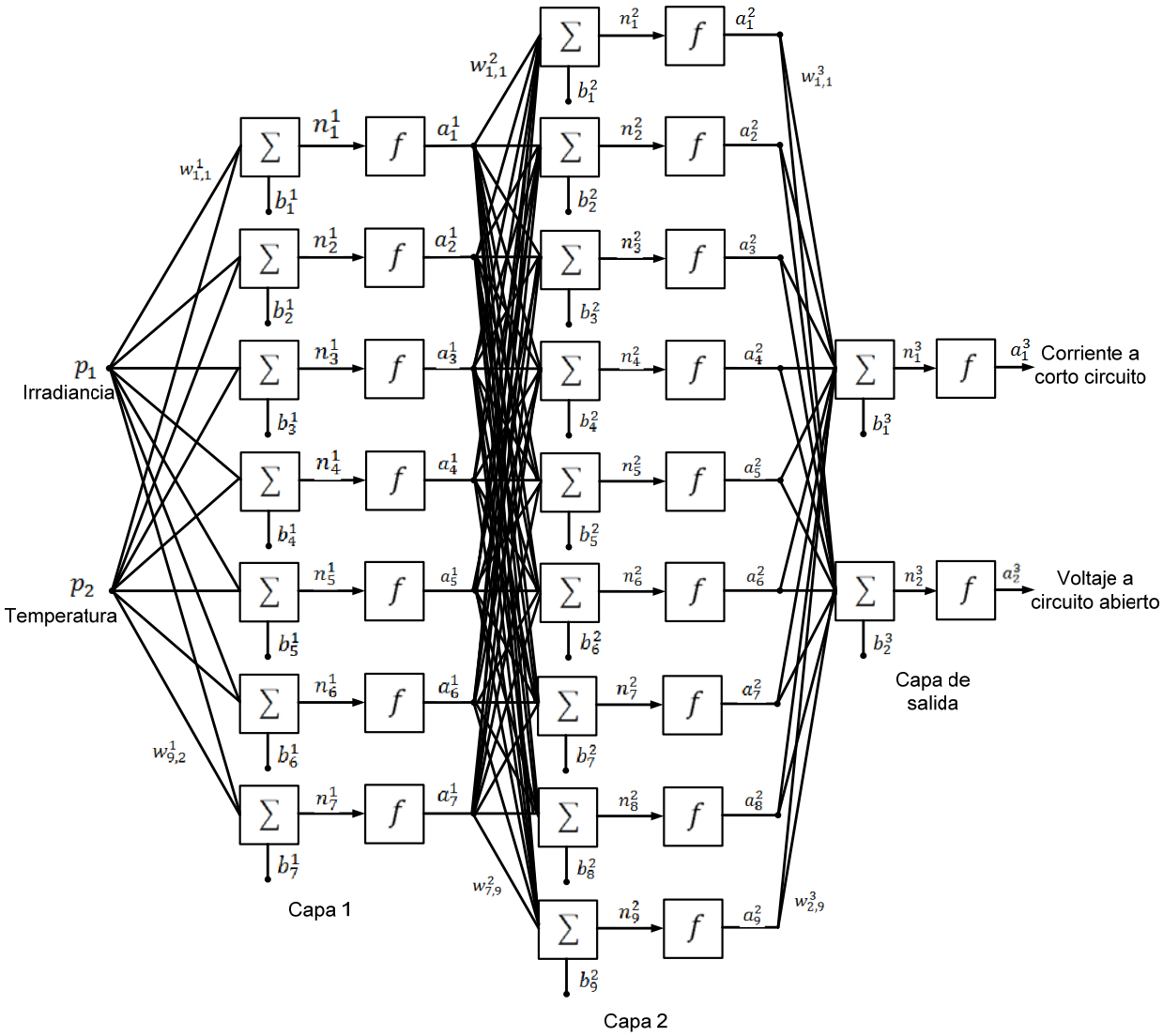


Fig. (3.1) *Arquitectura de la red neuronal multicapa.*

Esta arquitectura fue con la que se obtuvieron los mejores resultados durante el proceso de entrenamiento de la red neuronal, por lo cual en el Apéndice B, Sección 1, se puede observar el código que se implementó para la creación de dicha red. En la Fig. (3.2), se puede observar la red neuronal que se implementó en Matlab.

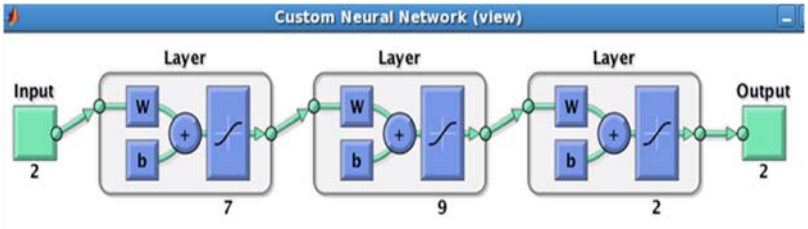


Fig. (3.2) *Red neuronal implementada en Matlab.*

3.6 Inicialización de los pesos.

Después de que se configuró la red, se realizó la inicialización de forma aleatoria de los parámetros ajustables (pesos y polarizaciones), esta inicialización se realizó en un intervalo de +/- 0.5.

3.7 Entrenamiento de la red.

Se realizó un entrenamiento de tipo *batch* (por lote), en el cual todos los pesos y las polarizaciones se actualizan después de que se le presentan todos los datos de entrada y salida objetivo a la red neuronal. Además de que se configuraron ciertos parámetros, como fueron: el algoritmo de entrenamiento Levenberg-Marquardt, utilizando la función "*trainlm*", el número de épocas o iteraciones que se desea entrenar la red, el valor mínimo que se quiere alcanzar de la función de desempeño (MSE) y un parámetro denominado *validation checks*, al cual se le asignó un valor de 6, este parámetro se va a utilizar para detener el proceso de entrenamiento, si el MSE del subconjunto de datos denominado validación se incrementa por 6 iteraciones consecutivas. Además del *validation checks* existen otros parámetros que van a detener el entrenamiento, como son: el número de épocas establecidas, el valor mínimo del gradiente y el valor de la función de desempeño (MSE). En la Fig. (3.3) se puede observar la interfaz que se obtuvo después de entrenar a la red neuronal donde se puede observar que la red neuronal se entrenó con 52 iteraciones y el proceso de entrenamiento se detuvo al cumplirse la condición *validation checks*, además de observar los valores del gradiente y del desempeño que se obtuvieron.

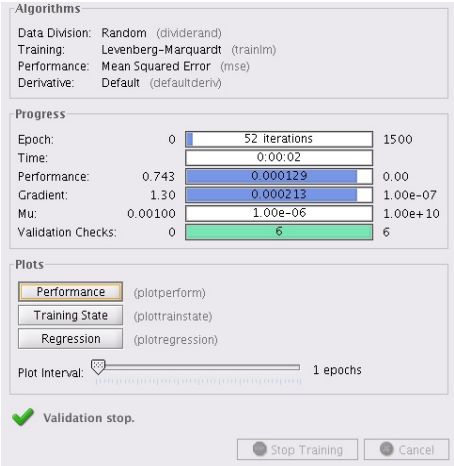


Fig. (3.3) *Interfaz de entrenamiento.*

3.8 Criterios de evaluación.

Una vez que se detuvo el proceso de entrenamiento de la red neuronal, se verificó que el entrenamiento que se realizó fuera el más óptimo. Esto quiere decir que el MSE que existe entre la salida de la red y la salida objetivo fuera el mínimo o lo que también significaría que se alcanzó un mínimo global de la función de error o de desempeño. Cuando el entrenamiento no es el más óptimo, existen varios métodos que se pueden realizar, como sigue.

- *Volver a inicializar los pesos y las polarizaciones, para realizar el entrenamiento nuevamente.*
- *Aumentar el número de neuronas o capas de la red.*
- *Incrementar el número de muestras de la base de datos.*

En nuestro caso, se tuvo que repetir la inicialización de los pesos y las polarizaciones, aproximadamente 20 veces hasta que se obtuvo el mínimo MSE, para determinar que el entrenamiento realizado fuera el mejor; se utilizaron dos graficas (*Error cuadrático medio* y *Gráfica de regresión*), las cuales están disponibles en el toolbox de redes neuronales.

3.8.1 Error cuadrático medio MSE.

La función de desempeño para las redes neuronales de tipo *feedforward*, es el error cuadrático medio (MSE), el cual es la diferencia cuadrática promedio entre la salida de la red neuronal y la salida objetivo a la que se quiere llegar; esta función de desempeño se puede representar con la ecuación (3.2).

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(e_i)^2 \qquad (3.2)$$

Substituyendo e_i por $(t_i - a_i)$ tenemos la siguiente expresión (3.3)

$$MSE = \frac{1}{N}\sum_{i=1}^{N}(t_i - a_i)^2 \qquad (3.3)$$

Donde:

N = Número total de las muestras.

t_i = Salida objetivo de la enésima muestra.

a_i = Salida enésima de la red neuronal.

Como se mencionó anteriormente, durante el proceso de entrenamiento, a la red neuronal se le presentaron dos subconjuntos de datos: (*Entrenamiento, Validación*), para la obtención de los pesos y las polarizaciones y una vez que finalizó el proceso de entrenamiento, se le presentó a la red neuronal el último subconjunto de datos (*prueba*), para observar el comportamiento ante datos que nunca observó durante el proceso de entrenamiento. De estos tres subconjuntos de datos se obtuvieron sus respectivos valores MSE, como se puede observar en la tabla (3.1). Como se puede ver, estos tres valores son muy cercanos al valor MSE = 0, que se estableció al ajustar los parámetros de entrenamiento, por lo que esto es un claro indicativo de que la diferencia que existe entre las salidas de la red neuronal con las salidas objetivo es mínima.

$MSE_{(TRAIN)}$	1.2585E-4
$MSE_{(VALIDATION)}$	1.4254E-4
$MSE_{(TEST)}$	1.7971E-4

Tabla (3.1)

En la Fig. (3.4) se puede observar la evolución del MSE de los tres subconjuntos de datos durante el proceso de entrenamiento, donde se puede apreciar cómo el MSE presenta un incremento en las primeras épocas de entrenamiento, pero a medida que el número de épocas aumenta, el MSE de los tres subconjuntos empieza a disminuir hasta llegar a los valores reportados en la tabla anterior.

En esta figura también se puede observar cómo el perfil que siguen los subconjuntos de validación y entrenamiento es muy parecido. Esto se debe a que estos dos subconjuntos están presentes durante el proceso de entrenamiento, caso contrario al perfil del subconjunto de prueba, que como podemos observar presenta un perfil diferente y un valor de MSE mayor a los dos subconjuntos anteriores.

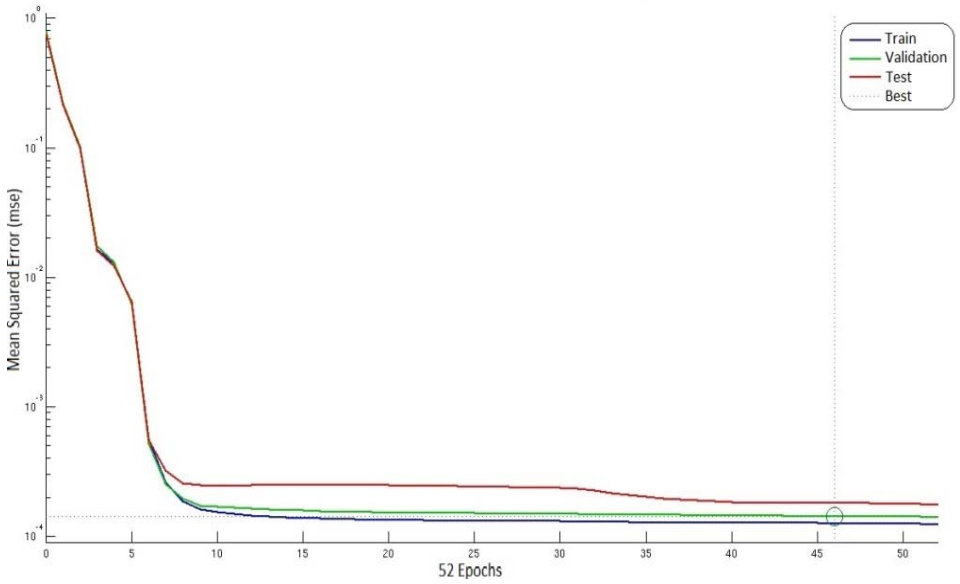


Fig. (3.4) Gráfica del error cuadrático medio, MSE

3.8.2 Gráfica de regresión.

Una forma de saber si los datos que se obtuvieron a la salida de la red neuronal son cercanos o iguales a los datos que se utilizaron como salida objetivo, es a partir del uso de dos herramientas estadísticas denominadas: *coeficiente de correlación de Pearson* y *línea de regresión lineal*, ya que existen situaciones en las que se requiere estudiar la relación entre dos o más variables, como es nuestro caso, (salida vs. salida objetivo). Para realizar el análisis de los datos nos apoyamos en el uso de los *diagramas de dispersión*.

3.8.3 Diagrama de dispersión.

La forma más sencilla para determinar si existe o no correlación entre dos variables es a partir de un diagrama de dispersión [4], en el cual se utiliza un sistema de coordenadas rectangulares XY, donde en el eje X (abscisas), se marca una escala adecuada para registrar los valores de una de las

variables y sobre el eje Y (ordenadas), se marca otra escala adecuada para registrar los valores de la otra variable, por lo que los valores de las variables forman pares ordenados (X, Y), dispersos en dicho sistema de coordenadas rectangulares.

3.8.4 Coeficiente de correlación de Pearson.

El coeficiente de correlación de Pearson (R) indicará el grado de relación que existe entre dos variables, o qué tanto depende una variable de otra, ya que se desea encontrar un valor numérico que exprese el grado de correspondencia o dependencia que existe entre las dos variables; este coeficiente es representado con la ecuación (3.4).

$$R = \frac{N\sum_{i=1}^{N} X_i Y_i - \sum_{i=1}^{N} X_i \sum_{i=1}^{N} Y_i}{\sqrt{N\sum_{i=1}^{N} X_i^2 - (\sum_{i=1}^{N} X_i)^2}\sqrt{N\sum_{i=1}^{N} Y_i^2 - (\sum_{i=1}^{N} Y_i)^2}} \qquad (3.4)$$

Donde:

N = número total de muestras

El valor de R va a ser un número que va a satisfacer la desigualdad $-1 \leq R \leq 1$.

R = +1, ocurre cuando la relación de las dos variables es perfectamente positiva, esto quiere decir que cuando se varía la primera variable, la segunda varía en la misma proporción y en la misma dirección.

R = -1, ocurre cuando la relación de las dos variables es perfectamente negativa, esto quiere decir que cuando se varía la primera variable, la segunda varía en las misma proporción pero en dirección contraria.

R = 0, ocurre cuando no existe relación entre las dos variables, esto es cuando al variar la primera variable las variaciones de la segunda no reflejan dependencia o conexión alguna con la primera variable.

3.8.5 Regresión Lineal.

El modelo estadístico de regresión lineal se utiliza para hacer deducciones o predicciones estadísticas, ya que este modelo nos va a permitir estimar el valor de una variable aleatoria (variable dependiente), dado que el valor de una variable asociada (variable independiente) es

conocido, por lo que se debe encontrar la recta de regresión, la cual también se le conoce como recta de mejor ajuste, ya que ésta nos va a permitir hacer predicciones y observar que para cada valor tabulado de (Y), existe un valor de predicción ($\hat{Y}$), que pertenece a la recta.

La recta de mejor ajuste se va a representar con la ecuación (3.5):

$$\hat{Y} = A + BX \tag{3.5}$$

Donde:

A = ordenada en el origen de la recta.

B = pendiente de la recta.

De la ecuación (3.5), se tienen que encontrar los valores de las constantes A y B; esto se realiza con las siguientes expresiones.

$$B = \frac{(1/N)\sum_{i=1}^{N} X_i Y_i - \bar{X}\bar{Y}}{(1/N)\sum N^2 - \bar{X}^2}$$

$$A = \bar{Y} - B\bar{X}$$

Donde:

$\bar{X}$ = media de X.

$\bar{Y}$ = media de Y.

Si la diferencia (Y - $\hat{Y}$) entre cada uno de los puntos tabulados y la recta es mínima, se tendrá un mejor modelo de predicción. Por lo que el cálculo de la recta de regresión se reduce a calcular los valores de A y B para los cuales el valor del error D es mínimo, lo que se representa con la ecuación (3.6).

$$D = \sum_{i=1}^{N} \left(Y_i - \hat{Y}\right)^2 = \sum_{i=1}^{N} [Y_i - (A + BX_i)]^2 \tag{3.6}$$

En la Fig. (3.5) se pueden observar los diagramas de dispersión que se obtuvieron para los tres subconjuntos de datos, donde las dos variables a representar en el sistema rectangular son: la salida de la red neuronal (eje Y) y la salida objetivo propuesta (eje X).

Se puede ver que en las tres graficas se tiene una correlación positiva, lo cual significa que al variar una variable, la otra varía en la misma proporción y en la misma dirección. Los valores de

correlación de Pearson que se obtuvieron se muestran en la tabla (3.2), donde se puede observar que el valor de *R* es muy cercano a la unidad en los tres casos, lo cual nos indica que existe una muy buena relación entre la salida y la salida objetivo o, en pocas palabras, que la salidas de la red neuronal se aproximan en gran medida a las salidas objetivo.

$R_{(\text{TRAIN})}$	0.99938
$R_{(\text{VALIDATION})}$	0.99953
$R_{(\text{TEST})}$	0.99879

Tabla 3.2

Además, se obtuvieron las ecuaciones de la recta de regresión o recta de mejor ajuste para los tres subconjuntos de datos, como se puede observar en la tabla (3.3). Esto permitirá hacer predicciones estadísticas, para comparar nuestras observaciones empíricas con dicho modelo.

Entrenamiento	$\hat{Y} = 1 * Target - 0.000038$
Validación	$\hat{Y} = 1 * Target + 0.00097$
Prueba	$\hat{Y} = 1 * Target - 0.00037$

Tabla 3.3

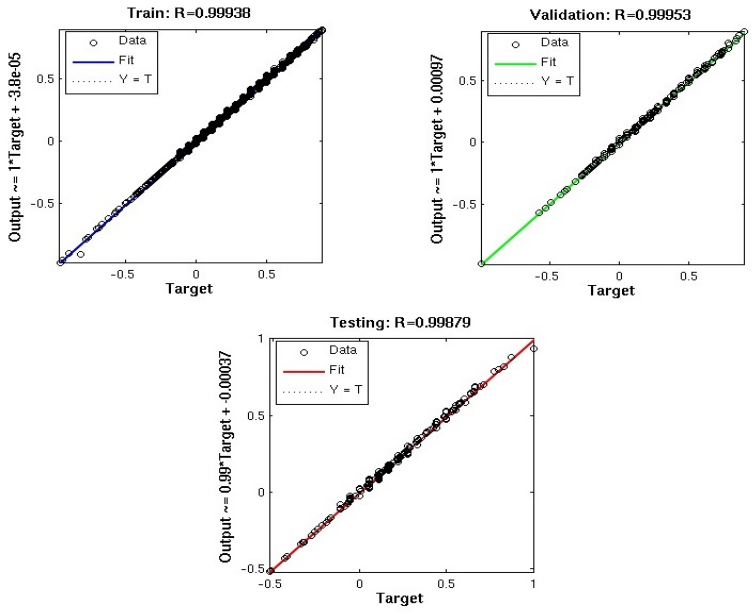


Fig. (3.5) Diagramas de dispersión.

3.9 Pesos y Polarización obtenidos.

Los valores de los pesos y polarización, van a ser de gran importancia ya que como se verá más adelante estos valores van a ser utilizados para realizar la implementación a nivel de hardware de la red neuronal propuesta, en un dispositivo digital FPGA, lo que va a permitir emular el comportamiento de un panel solar en tiempo real. Como se mencionó anteriormente, se obtuvieron tres matrices: [W^1_{7x2}, W^2_{9x7} y W^3_{2x9}] y tres vectores columna: [b^1_{7x1}, b^2_{9x1} y b^1_{2x1}], lo que arrojó un total de 95 pesos y 18 polarizaciones. Una vez que ya se validó el entrenamiento de la red neuronal, los valores de los pesos y de las polarizaciones que se obtuvieron fueron los siguientes:

Pesos de la primera capa oculta.

$$\boldsymbol{W^1} = \begin{bmatrix} 3.2644 & -0.1453 \\ 1.3366 & 2.6981 \\ -0.1921 & -1.0772 \\ 1.5404 & -0.5965 \\ -2.7475 & 0.8118 \\ 1.7215 & 1.8069 \\ 0.9812 & 2.9127 \end{bmatrix}$$

Pesos de la segunda capa oculta.

$$\boldsymbol{W^2} = \begin{bmatrix} -0.3398 & -0.2424 & 2.2110 & 0.1038 & 0.5535 & 0.7314 & -0.3163 \\ 1.1976 & 0.2422 & -0.1539 & 0.6816 & -1.3273 & 0.7897 & -0.6937 \\ 0.4269 & 0.4741 & 1.1026 & -0.0265 & -0.0091 & 0.0899 & -0.8289 \\ -1.2648 & 0.2356 & 0.3235 & 0.7467 & -0.7917 & 0.2571 & 0.9307 \\ -0.2772 & -0.0938 & -0.1995 & 0.2790 & -1.8310 & 0.3222 & 1.5726 \\ -0.0593 & 1.8922 & -1.7591 & -0.6109 & 0.4892 & -0.5084 & -0.3379 \\ 1.2958 & 0.4722 & -0.0612 & -1.3325 & 0.4674 & -0.1837 & -0.0084 \\ 0.7117 & 0.1056 & -0.3832 & 0.4173 & 0.0700 & -0.8756 & -0.4493 \\ 0.9564 & -0.3105 & 1.1809 & -0.0713 & -0.6539 & -0.8927 & 0.7837 \end{bmatrix}$$

Pesos de la capa de salida.

$$\boldsymbol{W^3} = \begin{bmatrix} -0.1135 & 2.6843 & -0.0339 & -0.0358 & 1.6786 & -0.1897 & -0.9104 & 0.9398 & -0.7300 \\ 0.8679 & -0.3211 & 1.0304 & 0.2659 & 0.0134 & -1.0425 & 0.1540 & 1.5162 & 0.7624 \end{bmatrix}$$

Polarizaciones obtenidas de la primera, segunda capa oculta y de la capa de salida.

$$\boldsymbol{b}^1 = \begin{bmatrix} -3.9429 \\ -2.8206 \\ -0.5546 \\ -0.6415 \\ -1.9399 \\ 2.2629 \\ 3.8237 \end{bmatrix} \qquad \boldsymbol{b}^2 = \begin{bmatrix} 2.1250 \\ -1.3212 \\ 0.8750 \\ 0.5789 \\ 0.0269 \\ -1.2140 \\ 0.6450 \\ -2.6055 \\ 1.7667 \end{bmatrix} \qquad \boldsymbol{b}^3 = \begin{bmatrix} 1.5985 \\ -0.4867 \end{bmatrix}$$

Conclusiones.

En este capítulo se revisó el proceso que se llevó a cabo para la implementación de una red neuronal multicapa de tipo feed-forward, en el ambiente de desarrollo de Matlab. Se realizó la creación de una red neuronal con una arquitectura (2-7-9-2), la cual se entrenó utilizando el algoritmo de aprendizaje de Levenberg-Marquardt, para así de esta manera obtener los valores de los pesos y de las polarizaciones óptimas que servirán para la implementación de esta red neuronal en un dispositivo digital FPGA.

Se verificó a partir de la gráfica del error cuadrático medio (MSE), que durante el proceso de entrenamiento este error presentó una disminución, hasta que la diferencia entre la salida objetivo y la salida fuese mínima, cercana a cero.

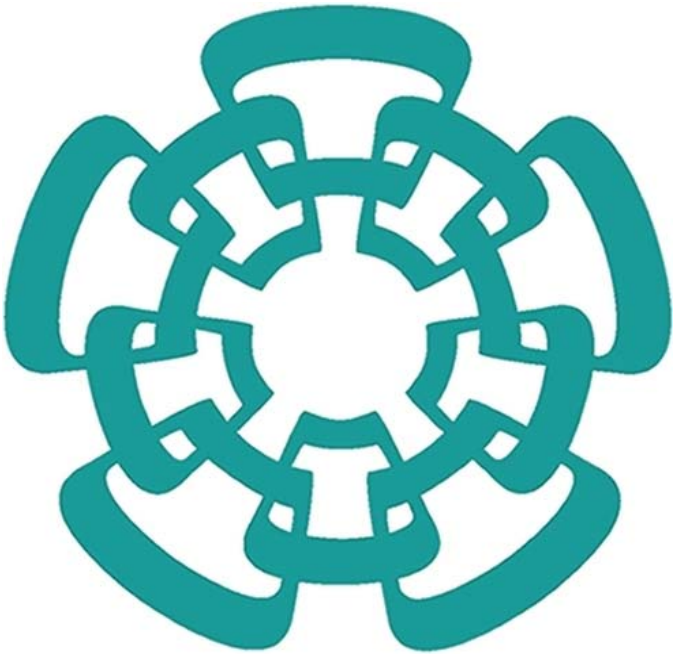
También se observó a partir de la gráfica de regresión, que existió un buen grado de relación entre la salida de la red con la salida objetivo, ya que el coeficiente de Pearson (R) se aproximó a la unidad, en los tres subconjuntos de datos, lo cual indicó que el entrenamiento de la red neuronal fue el más óptimo. El toolbox de redes neuronales de Matlab fue de gran ayuda en la creación y entrenamiento de nuestra red neuronal.

Referencias.

1. M. Hagan, H. Demuth y M. Beale, “Neural Network Design” Oklahoma State University.
2. L. Fausett, “Fundamentals of Neural Networks Architectures, Algorithms and Applications”
3. Neural Network Toolbox Matlab.
4. A. Naiman, R. Rosenfeld y G. Zirkel, “Introducción a la Estadística” México, D.F. Editorial Mc Graw Hill 1987.

CAPÍTULO 4

Implementación de una red neuronal en VHDL



4.1. Introducción.

En el capítulo anterior se realizó el entrenamiento de una red neuronal artificial, así como la obtención de los valores de los pesos sinápticos y polarizaciones para cada neurona. Por lo que en este capítulo se presentarán los procedimientos que se siguieron para la implementación de dicha red neuronal, en un dispositivo digital FPGA (*Field Programmable Gate Array*), utilizando el lenguaje de descripción de hardware VHDL.

En este capítulo se describirá la creación de cada módulo funcional que conforma a una neurona digital, para de esta manera, realizar la unión de varias neuronas digitales, para conformar, la red neuronal descrita en el capítulo anterior. Además, se realizará la descripción de un módulo controlador, el cual permitirá la generación de señales de control, las cuales serán de importancia para la sincronización y el control del flujo de datos, entre las neuronas de cada capa, así como también se explicará la creación de otros módulos que permitirán almacenar los datos de entrada y salidas en bloques de memoria de tipo RAM. También se mostrará un diagrama a bloques donde se podrán apreciar módulos que serán utilizados para realizar la transmisión de nuestros datos de salida hacia el exterior del FPGA, para su análisis posterior.

Para la descripción de los módulos que conforman a una neurona digital y de la misma red neuronal, se utilizó la herramienta de desarrollo *ISE Design Suite de Xilinx*, en su versión 14.2, además de que todas las simulaciones que se presentan en este capítulo fueron realizadas con el programa *ModelSim* de *ModelTech* en su versión 10.0c. El dispositivo digital utilizado en este trabajo fue el *FPGA Spartan-6 LX45* de *Xilinx*, modelo CSG324C, el cual a su vez se encuentra implementado en una tarjeta de desarrollo modelo Atlys.

4.2 Procesamiento de datos (analógico / digital).

Como se recordará, en el Capítulo 3 se realizó la normalización de la base de datos meteorológica: (Irradiancia y Temperatura), dentro de un rango de +/- 1, para poder realizar el proceso de entrenamiento de la red neuronal artificial en Matlab, además de la división de la base de datos en 3 subconjuntos denominados: *Entrenamiento, Validación y Prueba*.

En este capítulo se establecerá el subconjunto denominado "*Prueba"*, el cual como se recordará, está conformado por 165 valores de Irradiancia y 165 valores de Temperatura, seleccionados aleatoriamente. Debido a que se busca realizar la implementación de la red neuronal en un dispositivo digital, los valores de entrada, así como los valores de los pesos sinápticos y polarizaciones obtenidos en el entrenamiento, deben ser convertidos de un formato analógico a un formato digital.

Este proceso consiste en representar un dato analógico con una palabra digital de un tamaño de 18 bits, donde con 9 bits (bit 18 al bit 10), se representará la parte entera del valor analógico y con los 9 bits restantes (bit 9 al bit 1) se representará la parte fraccionaria. Por ejemplo, para representar el valor analógico 3.25, su representación formato digital será la siguiente:

$$000000011.010000000_2$$

Parte Entera — Parte Fraccionaria

Para evitar el uso del punto flotante en la descripción realizada en VHDL, se trabajaron todas las palabras con un tamaño de 18 bits, como números enteros, eliminando de esta manera el punto decimal en cada valor, como se muestra a continuación:

$$000000011010000000_2 = 1664_{10}$$

Debido a que se tienen algunos valores negativos de pesos y polarizaciones, como se puede observar en el Capítulo 3, para la conversión analógica digital de estos valores, se realizó el complemento a dos, ya que al tener el bit más significativo con un valor de '1', el sintetizador del simulador de ModelSim, interpreta este valor como negativo. Esto se puede ver utilizando el valor anterior pero ahora negativo -3.25.

El valor obtenido sin complemento a dos es el siguiente:

$$000000011010000000_2$$

Al aplicar el complemento a dos se obtiene:

$$111111100110000000_2 = 260480_{10}$$

Esta es la manera con la que se manejarán todos los datos digitales. Así siempre se debe tener en cuenta que los resultados binarios que resulten en las simulaciones, estarán representados de la manera que acaba de ser descrita.

4.3 Red neuronal digital Simple.

Como ya se mencionó anteriormente, el objetivo principal de este capítulo es el de implementar dentro de un dispositivo FPGA, la arquitectura de la red neuronal artificial que se describió en el Capítulo 3, Fig. (3.1), por lo que se realizó el siguiente diagrama esquemático básico, Fig. (4.1). Como se puede apreciar en el diagrama, se tienen dos variables: una denominada (H), la cual es usada para hacer referencia a los valores de la Irradiancia y una variable (T) para los valores de la Temperatura: También se puede observar que se emplea una variable (I) para representar a los valores de la corriente a corto circuito y otra variable denominada (V) para los valores del voltaje a circuito abierto. Asímismo, se puede observar que el tamaño de los ductos de información con los que se trabajó en este diagrama son todos de 18 bits como se explicó anteriormente.

En este diagrama también se aprecian bloques que van a representar el número de neuronas digitales que se van a tener en cada capa oculta: 7 para la primera capa y 9 para la segunda capa oculta, así como 2 neuronas en la capa de salida. Estos bloques se explicarán con más detalle más adelante. Finalmente se implementan 3 multiplexores, los cuales cumplirán con la función de permitir el paso de la información a través de las capas, como se verá más adelante.

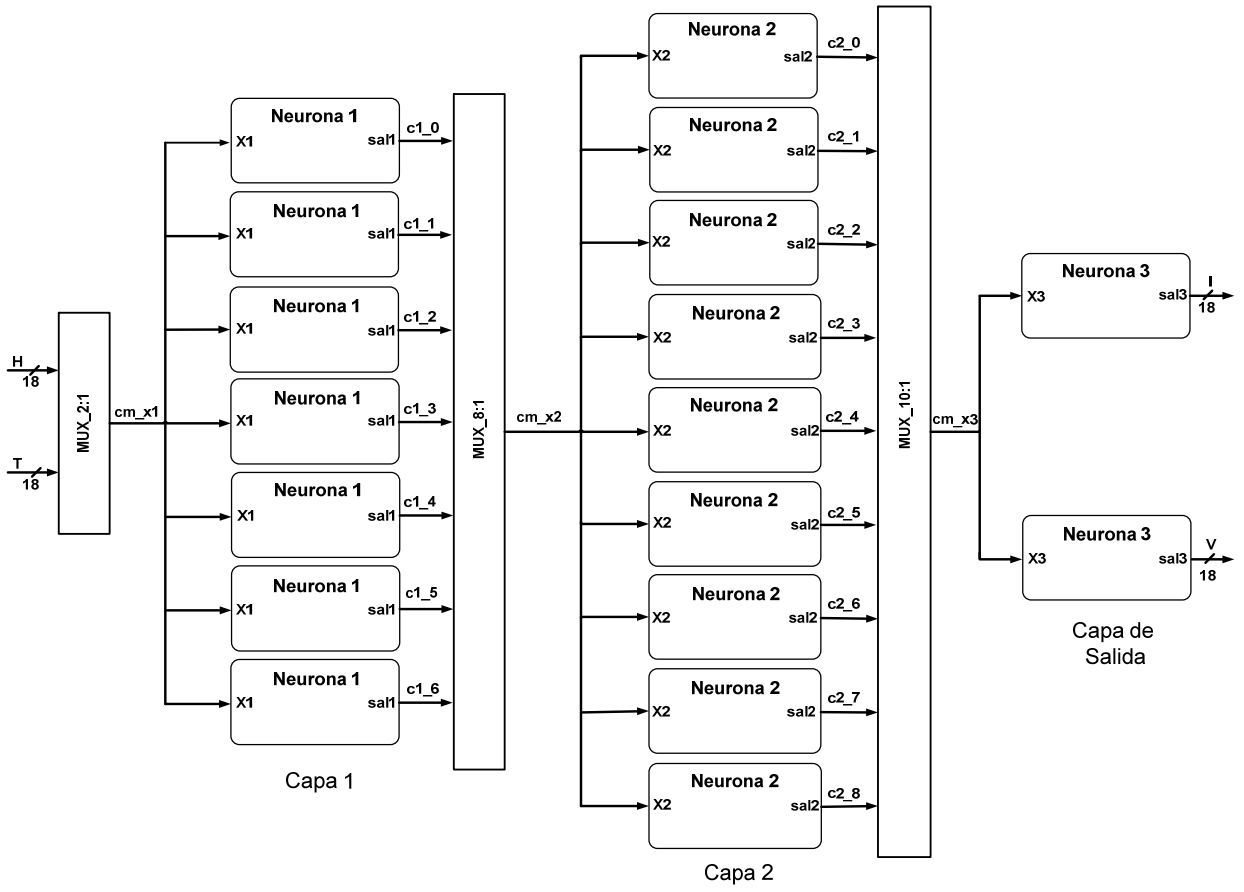


Fig. (4.1) *Diagrama esquemático básico de la red neuronal digital.*

4.4 Diseño de una neurona digital.

Como se ve en el diagrama anterior y como se presentó en capítulos anteriores, el componente fundamental de cualquier red neuronal artificial, es la unidad de procesamiento o neurona, ya que ésta es la que se encarga de procesar la información de entrada, para así obtener un valor de salida. Como se recordará, dentro de una neurona se realizan varios procesos: el producto de las entradas con los pesos sinápticos, la suma y acumulación de las entradas ponderadas con una polarización y la generación de la salida de la neurona a través de una función de activación.

Todos estos procesos serán descritos usando el lenguaje de descripción de hardware VHDL, como se podrá observar en las secciones siguientes. En la Fig. (4.2a) se muestra a la entidad de nivel superior **Neurona_1** de la capa 1 que se utilizó en la Fig. (4.1) y en la Fig. (4.2b) podemos observar con mayor detalle cómo esta entidad de nivel superior se encuentra conformada por

varias entidades de nivel inferior, las cuales nos van a permitir realizar todos los procesos que se efectúan dentro de una neurona artificial.

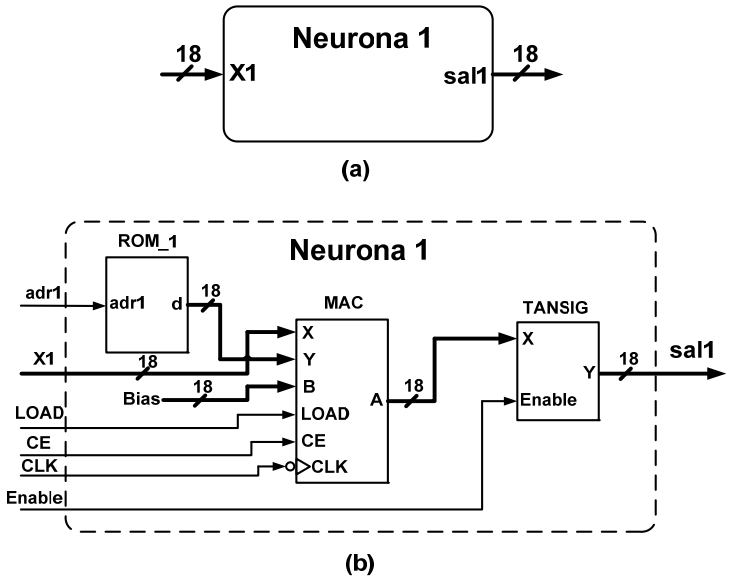


Fig. (4.2) *(a) Diagrama a bloques simple de una neurona digital. (b) Diagrama a bloques completo de una neurona digital.*

4.4.1 Unidad de Almacenamiento ROM_1.

Como se puede observar en la Fig. (4.2b), la entidad Neurona_1 va a tener un bloque denominado **ROM_1**, el cual va a servir para almacenar y transmitir los valores de los pesos obtenidos en el Capítulo 3, para cada neurona digital de las distintas capas. También vale la pena mencionar que la arquitectura utilizada para este bloque no es la de una memoria de solo lectura (ROM), únicamente se utilizó esta denominación para hacer referencia a la entidad.

En la Fig. (4.3a) se muestra a la entidad que fue descrita en lenguaje VHDL para la unidad de almacenamiento ROM_1. Como se puede apreciar en esta misma figura, esta entidad únicamente se compone por un puerto de entrada denominado **adr1** y un puerto de salida denominado **d**. Además, en la Fig. (4.3b), se puede observar el tamaño de la entidad ROM_1, en este caso se tienen dos constantes de tipo entero denominadas **X1** y **X2**, las cuales van a servir para almacenar los valores de los pesos $\mathbf{W_{11}}$ y $\mathbf{W_{12}}$ de la neurona 1. Esta misma entidad se utilizó en las 7 neuronas que conforman a la capa 1 de la red neuronal digital debido a que el tamaño de la unidad de almacenamiento se encuentra en función del número de entradas que se tengan, por tal

motivo las unidades de almacenamiento **ROM_2** y **ROM_3** utilizadas en las entidades **Neurona_2** y **Neurona_3** no van a tener el mismo tamaño como se verá más adelante.

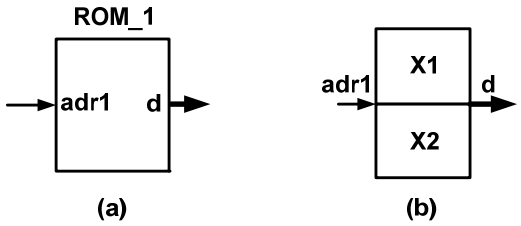


Fig. (4.3) *(a) Unidad de almacenamiento ROM_1 de la neurona de la capa 1. (b) Tamaño de la unidad de almacenamiento ROM_1.*

A continuación, en el listado 4.1 se muestra parte del código utilizado para la descripción del comportamiento de la entidad ROM_1, donde como se puede apreciar, el funcionamiento de esta entidad se asemeja al de un Flip-Flop tipo D, ya que a partir del valor binario (1 ó 0) que se tenga en la señal de control **adr1,** se va a transferir ya sea el valor del peso sináptico almacenado en la constante **X1** o el almacenado en **X2**, al puerto de salida **d**. El código completo para esta entidad se puede ver en el Apéndice D, Sección 1, Listado (1).

```
26    architecture Behavioral of ROM_1 is
27    begin
28      process (adr1)
29       begin
30        case adr1 is
31          when '0' => d <= CONV_std_logic_vector(x1,Size);
32          when others => d <= CONV_std_logic_vector(x2,Size);
33        end case;
34    end process;
35    end Behavioral;
```

Listado 4.1*. Descripción de la entidad ROM_1.*

4.4.2 Diseño de la unidad MAC (Multiply – Accumulate).

La siguiente entidad que se desarrolló fue la que lleva por nombre **MAC,** Fig. (4.4). Esta entidad fue empleada para realizar las siguientes operaciones, las cuales son fundamentales en las neuronas artificiales.

- Ponderación de los valores de entrada realizando su multiplicación con los pesos sinápticos.

- Acumulación de las entradas ponderadas.
- Suma del valor de la polarización al valor obtenido de la acumulación.

La entidad que se desarrolló en VHDL se puede observar en la Fig. (4.4a); esta entidad se encuentra conformada por 6 puertos de entrada, de los cuales 3 puertos serán utilizados para el ingreso de datos (X, Y, B) y los 3 puertos restantes se utilizarán para el ingreso de señales de control (LOAD, CE, CLK) además se tendrá además un puerto de salida (A).

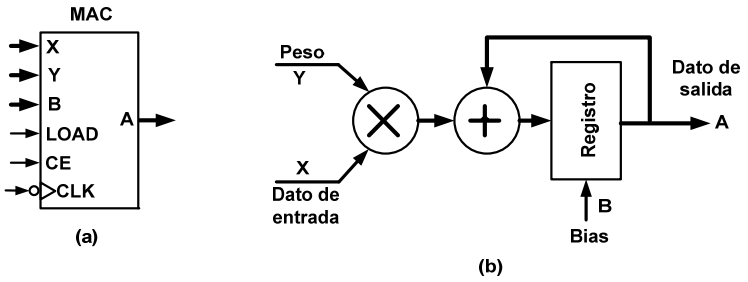


Fig. (4.4) *(a) Entidad MAC. (b)Estructura interna de la Entidad MAC.*

En el siguiente listado (4.2) se explican algunas partes del código de la entidad MAC, el cual puede ser consultado en el Apéndice D, Sección 1, Listado (2).

```
21    XY <= X*Y;
22    A <= ACC;

24    process(CLK,LOAD,ACC,XY,BIAS,CE)
25          begin
26          if falling_edge (CLK) and (CE='1') then
27                if (LOAD = '1') then
28                      ACC <= BIAS;
29                else
30                      ACC <= ACC + XY;
31                end if;
32          end if;
33    end process;
34    end Behavioral;
```

Listado (4.2) *Descripción del comportamiento de la entidad MAC.*

En la línea 21, se puede apreciar la multiplicación entre un valor de entrada **X** con un peso sináptico **Y**, algo que es de suma importancia aclarar, ya que para realizar este tipo de operaciones se utilizó la multiplicación directa del lenguaje VHDL, debido a que este

multiplicador acepta dos operandos con un mismo tamaño de palabra de 18 bits; por tal motivo no fue necesario realizar el diseño de un nuevo multiplicador.

Una señal de control que es de suma importancia es la denominada **LOAD**, ya que como se puede ver en la línea 27, cada vez que tenga un valor igual a '1', permite cargar el valor de la polarización en una variable denominada **ACC** que cumplirá la función de ir acumulando los resultados obtenidos, línea 28; en el caso contrario de que la señal **LOAD** presente un valor binario de '0', lo que se realizará, como se puede ver en la línea 30, será la suma del producto realizado con la polarización. Además cabe mencionar que se está utilizando una señal de de reloj, la cual se operó a una frecuencia de 10 MHz.

4.4.3 Función de Transferencia TANGSIG.

Otro bloque que se desarrolló para la creación de la neurona digital Neurona_1, fue el denominado **TANSIG**, el cual representa a la función de activación que se va a usar. En este caso, como se ha explicado anteriormente, se utilizará la función tangente sigmoidal o tangente hiperbólica; en la Fig. (4.5b) se puede observar su representación gráfica.

La entidad que se creó para este bloque, se muestra en la Fig. (4.5a), en la cual se puede observar que se van a tener dos puertos de entrada: uno para el ingreso del valor de salida obtenido de la entidad MAC puerto (**X**) y otro, puerto de entrada (**Enable**), para permitir el paso del resultado obtenido al evaluar la función tangente sigmoidal, hacia el puerto de salida denominado (**Y**), para así de esta manera obtener la salida de la neurona digital.

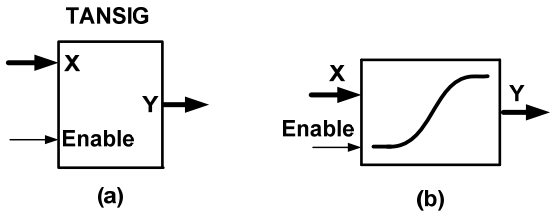


Fig. (4.5) *(a) Entidad de la función de activación* ***TANSIG****. (b) Representación grafica de la función Tangente Sigmoidal.*

Debido a que este bloque es de suma importancia, ya que permitirá desarrollar de forma digital la respuesta de la función tangente hiperbólica, se explicará a continuación gran parte del

comportamiento de dicha entidad, El código completo se puede consultar en el Apéndice D, Sección 1, Listado (3).

Antes de empezar a describir el código descrito en VHDL, se debe mencionar que para el desarrollo de la función tangente sigmoidal se utilizó el método de interpolación lineal, esto es, a partir de una serie de líneas rectas de la forma $y = ax + b$, se va aproximando a la función sigmoidal utilizando la pendiente (a) y la ordenada en el origen (b) de cada línea recta.

Para la implementación en lenguaje VHDL se utilizaron 16 segmentos de recta como se podrá ver más adelante.

4.4.3.1 Declaración de constantes: pendiente, ordenada en el origen, abscisas.

En la líneas (14 – 16) se muestran algunos de los valores de las pendientes (**a0 – a5**) que se obtuvieron de los 16 segmentos de rectas. Como se puede observar, están representados en un formato de tipo entero, el cual es resultado de la conversión de un número real a un valor digital de 18 bits. De la misma manera, se puede observar la representación en formato entero de algunos de los valores obtenidos para las ordenadas en el origen (**b0 – b5**), líneas (23 - 25). Para consultar la representación numérica de las constantes descritas anteriormente, consultar el Apéndice D, Sección 6, Listado (1). Además de tener el valor de la pendiente y la ordenada en el origen, se realizó la segmentación del eje de las abscisas, de la parte positiva de la función de activación en 17 partes (**x0 – x5**), dentro de un rango de 0 a 3.5, líneas (32 – 40). Estos valores serán de utilidad para realizar la selección del segmento de recta a evaluar, como se podrá ver más adelante.

```
14    constant a0:integer:=508; constant a1:integer:=473;
15    constant a2:integer:=414; constant a3:integer:=342;
16    constant a4:integer:=270; constant a5:integer:=205;

23    constant b0:integer:=0;   constant b1:integer:=7;
24    constant b2:integer:=29;  constant b3:integer:=70;
25    constant b4:integer:=124; constant b5:integer:=185;

32    constant x0:integer:=0;    constant x1:integer:=96;
33    constant x2:integer:=192;  constant x3:integer:=288;
34    constant x4:integer:=384;  constant x5:integer:=480;
```

4.4.3.2 Selección de la recta a evaluar.

A continuación se explicará el modo de operación de la entidad. Primeramente se tuvo que obtener el valor absoluto del valor de entrada (línea 51). Esto se debe principalmente a que se segmentó únicamente la parte positiva de la función de activación, como se acaba de explicar. Una vez que el valor de entrada (**abs_x**) es positivo va a entrar a un proceso en el cual a partir de una serie de enunciados de tipo ***if – then – elsif – then – else*** y utilizando los puntos obtenidos de la segmentación del eje de las abscisas, se seleccionará el intervalo en el que se encuentra el valor de entrada, para de esta manera, obtener el valor de la pendiente y de la ordenada en el origen de la recta que se va a evaluar, líneas (56 – 106).

```
51    abs_x<=abs(X);

53    process(abs_x)
54    begin

56          if (abs_x>= X0)and(abs_x< X1) then
57                a<=conv_std_logic_vector(a0,18);
58                b<=conv_std_logic_vector(b0,18);
59          elsif (abs_x< X2) then
60                a<=conv_std_logic_vector(a1,18);
61                b<=conv_std_logic_vector(b1,18);
62          elsif (abs_x< X3) then
63                a<=conv_std_logic_vector(a2,18);
64                b<=conv_std_logic_vector(b2,18);
.
.
.
.
.
108   end process;
```

4.4.3.3 Evaluación de la recta.

Una vez que ya se obtuvieron los parámetros de la recta, se utilizó la ecuación de la recta, (línea 110), para de esta manera obtener el valor de la recta, el cual será el resultado de la función tangente sigmoidal. Este valor se le asignó a la variable **abs_y**, la cual va a tener un tamaño de 36 bits debido a la multiplicación que se realiza entre la pendiente y el valor de entrada, ambas con un tamaño de 18 bits.

Como se puede ver en las líneas 112 – 119, este valor entra a un proceso donde, a partir de un enunciado de tipo ***if – then – else***, se pregunta si el valor del bit más significativo (MSB) de nuestro valor de entrada **X(17)** es '1', lo cual significaría que el resultado obtenido en la variable

abs_y (26 downto 9), debe de ser negativo. Este resultado, como se observa en la línea 115, se le asigna a la variable **Y_S**, la cual tendrá un tamaño de 18 bits, en caso contrario si el bit más significativo es '0', el resultado será positivo, (línea 117).

Finalmente, cuando la señal de control **Enable** presente un valor de '1', el resultado se le asignará a la variable **latch**, la cual a su vez se le asignará al puerto de salida **Y**; en caso contrario, el resultado se mantendrá almacenado en la variable latch. Esto se realiza principalmente para asegurar que a la salida de la entidad se tendrá un valor estable.

```
110   abs_y<=(a*abs_x)+(b&O"000");
112   process(X,abs_y)
113   begin
114         if X(17)='1' then
115               Y_S <= O"000000"-abs_y(26 downto 9);
116         else
117               Y_S <= abs_y(26 downto 9);
118         end if;
119   end process;

121   latch <= Y_S when(Enable = '1') else latch;
122   Y <= latch;
124   end Arq;
```

4.4.3.4 Respuesta de la función de transferencia TANSIG.

En la figura siguiente se muestra la respuesta obtenida de la función de transferencia tangente sigmoidal, ante una señal de entrada de tipo rampa, la cual presenta un rango de valores de -3 a +3; se puede observar cómo la señal de salida se asemeja a la función de activación descrita en el Capítulo 1, además de tener la característica de saturar la salida en un valor de +/-1.

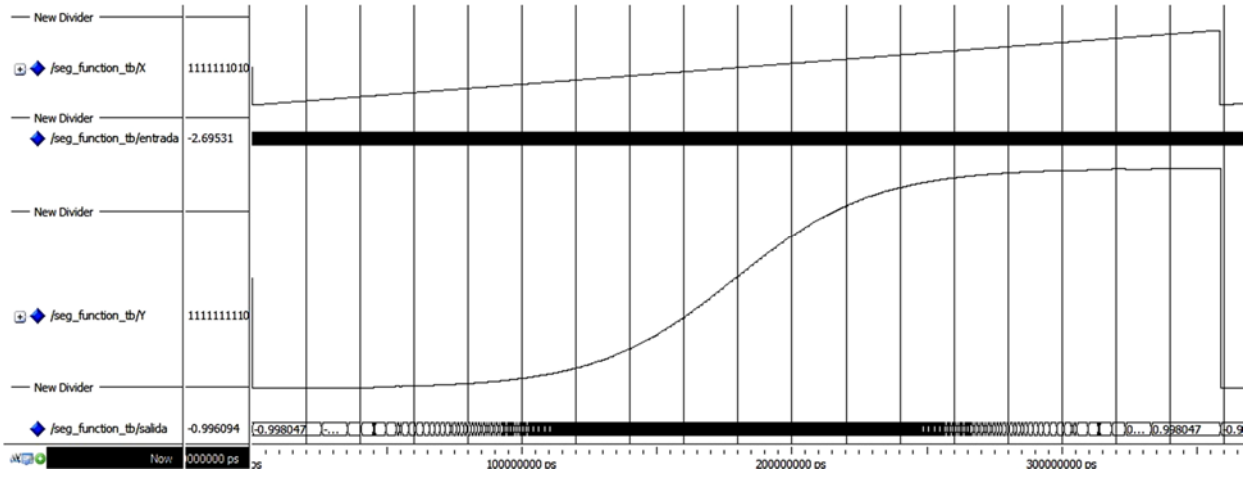


Fig. (4.6) *Resultado de la simulación de la función de transferencia TANSIG*

4.5 Diseño de una neurona digital.

Una vez que se han desarrollado y analizado todos los módulos que van a conformar a la neurona digital, es necesario integrarlos en una sola estructura denominada Neurona_1. En el siguiente listado podemos apreciar, cómo se realizó el llamado de los componentes ROM_1, MAC y TANSIG para integrarlos en la estructura de nivel superior Neurona_1. Para consultar el código completo, ver el Apéndice D, Sección 2, Listado (1).

```
28    -- Llamado de los componentes

29    U1:ROM_1
30    generic map(x1=>W1, x2=>W2, size=>18)
31    port map (d=>W, adr1=>adr1);

33    U2:MAC port map (X=>X1, Y=>W, B=>B, LOAD=>LOAD, CLK=>CLK, CE=>CE, A(26
34    downto9)=>A,A(35downto27)=>E(8 downto 0),A(8 downto 0)=>E(17downto 9));

36    U3:TANSIG port map (X=>A, Enable=>Enable, Y=>sal1);
37     end Behavioral;
```

Listado (4.3) *Llamado de componentes para conformar a la neurona 1.*

4.6 Diseño de una Red Neuronal Digital Simple.

Una vez que se creó y analizó la respuesta de una neurona digital, en este caso a la **Neurona_1**, el siguiente pasó que se realizó para la creación de la red neuronal digital simple descrita en la Fig. (4.1), fue la elaboración de las entidades **Neurona_2** y **Neurona_3**, además de las entidades de los multiplexores: **Mux_2:1**, **Mux_8:1** y **Mux_10:1**, como lo explicaremos a continuación.

4.6.1 Diseño de la entidad Neurona_2.

Esta entidad, se va a encontrar conformada por los mismos bloques que conformaron a la Neurona_1, pero el único bloque que será diferente, será el de la unidad de almacenamiento ROM_2, ya que como se explicó anteriormente, el tamaño de este bloque está en función del número de entradas que se tengan, en este caso, al ser una neurona de la capa 2, se tendrán 7 entradas. Por lo tanto, esta unidad de almacenamiento almacenará en su totalidad 7 pesos. En el Apéndice D, Sección 3, Listado (1), se muestra a la entidad de la unidad de almacenamiento ROM_2 y en el Listado (3), se muestra también el llamado de todos estos componentes para la formación de la Neurona_2. Una vez que se conformó la entidad de la Neurona_2, se hizo llamar 9 veces, para conformar a la capa 2.

4.6.2 Diseño de la entidad Neurona_3.

La descripción de esta entidad, igual que las anteriores, se muestra en el Listado (4) y en el Listado (2) se muestra a la entidad ROM_3, la cual va a estar conformada por un tamaño de 9. De la misma manera que en las neuronas anteriores, una vez que se conformó la entidad de la Neurona_3, se hizo su llamado 2 veces, para así de esta manera, conformar a la capa de salida de la red neuronal digital.

4.6.3 Multiplexor 2:1.

Este multiplexor, como se puede observar en la Fig. (4.1), va a servir para seleccionar la variable de entrada que va a ser procesada por las 7 neuronas de la capa 1. Esta selección se va a realizar por medio de una estructura ***with – select – when*** y una señal de control **SEL1**. Si esta señal presenta un valor de '0', a la salida del multiplexor se tendrá un valor de Irradiancia (H), en caso contrario con un '1', se tendrá un valor de Temperatura (T). Ver el Apéndice D, Sección 3, Listado (5) para la descripción en lenguaje VHDL.

4.6.4 Multiplexor 8:1.

Este multiplexor servirá para conectar la salida de cada neurona de la capa 1, con las 9 neuronas de la capa 2. Para esto se tendrá una señal de control denominada **SEL2**, la cual permitirá realizar dicha conexión, este multiplexor igual que el anterior utilizará para su descripción una estructura ***with – select – when***. Ver el Apéndice D, Sección 3, Listado (6) para la descripción en lenguaje VHDL.

4.6.5 Multiplexor 10:1.

Finalmente, se hará uso de un multiplexor de 10 entradas y una salida para realizar la conexión entre las salidas de las 9 neuronas de la capa 2 con las 2 neuronas de la capa de salida. Por medio de una señal de control denominada **SEL3** y utilizando la misma estructura ***with – select – when*** que en los multiplexores anteriores. Ver el Apéndice D, Sección 3, Listado (7) para la descripción en lenguaje VHDL.

4.6.6 Diagrama de tiempo de las señales de control.

En la Fig. (4.7) se muestra el diagrama de tiempos que se utilizó para el funcionamiento de la red neuronal digital simple. Como se puede ver, el diagrama va a estar dividido en 3 secciones: la

primera sección servirá para controlar a las 7 neuronas que conforman a la capa 1. En esta misma sección se emplea la señal SEL1, la cual como se vió anteriormente, será la llave de selección para el multiplexor 2:1, pero además esta misma señal se conectará a la señal de control adr1 de la unidad de almacenamiento ROM_1, para transmitir a la salida los valores de los pesos que se encuentran almacenados en dicha unidad. Otra señal de gran importancia es la denominada CE1, debido a que con esta señal se puede activar o desactivar el proceso de multiplicación y acumulación que se realiza dentro de la unidad MAC en todas las neuronas, ya que como se muestra en el tiempo **t0**, al presentarse un flanco de bajada en la señal de reloj CLK y CE1, se encuentra con un valor de '1' y además la señal LOAD tiene un valor de '1'. Esto permitirá cargar el valor de la polarización en el acumulador (ACC) de la unidad MAC de las 7 neuronas. Pero si ahora se presenta un flanco de bajada en la señal de reloj, CE = 1 y SEL1 = 0 como se puede apreciar en el tiempo t2, se realizará en las 7 neuronas la multiplicación entre el valor de la Irradiancia (H) obtenido del multiplexor 2:1 con el primer valor de peso sináptico X1, que se almacenó en la unidad ROM_1 y este resultado se sumará al acumulador (ACC). De la misma manera, esto ocurrirá en el tiempo t3, pero ahora la señal SEL1 tendrá un valor de '1', permitiendo de esta manera realizar la multiplicación entre el valor de la Temperatura y el valor de peso sináptico X2 y se realizará nuevamente la suma de este resultado con el valor del acumulador; finalmente, en el tiempo t4, CE1 cambia a un valor de '0' desactivando de esta manera el proceso de multiplicación y acumulación de la unidad MAC, lo cual significa que ya no hay mas pesos almacenados en la unidad ROM_1 que se puedan multiplicar con las entradas. Este resultado final será procesado por la función de activación y cada vez que la señal Enable presente un valor de '1', se transmitirá el valor resultante de la función de activación hacia la salida de cada neurona. Este mismo principio básico de operación se utilizó para controlar a las 9 neuronas que conforman a la capa 2 y a las 2 neuronas de la capa de salida.

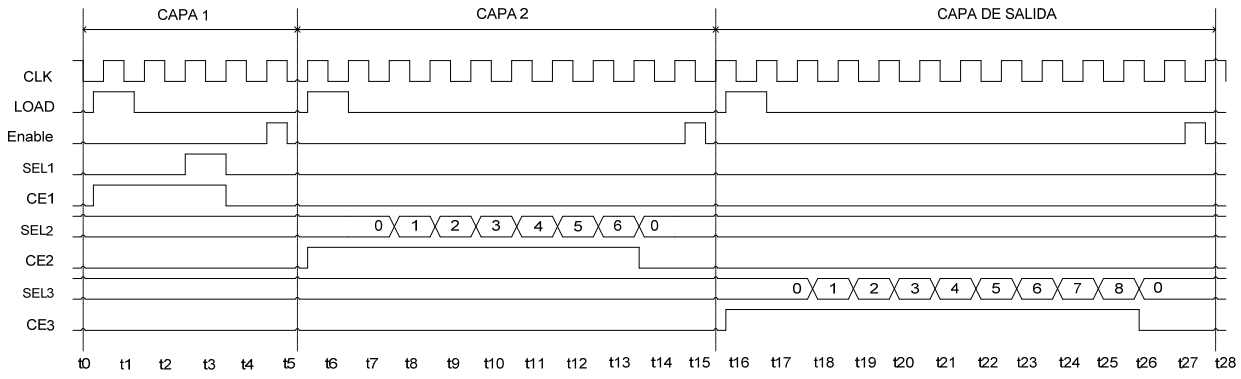


Fig. (4.7) *Diagrama de tiempo de las señales de control para la red neuronal simple.*

Una vez que se crearon todos los módulos descritos anteriormente se realizó la integración todos estos módulos en una entidad de mayor nivel jerárquico denominado **Red_Neuronal**; para consultar el llamado de todos los componentes consultar el Apéndice D, Sección 3, Listado (8).

4.6.7 Respuesta de una Red Neuronal Digital Simple.

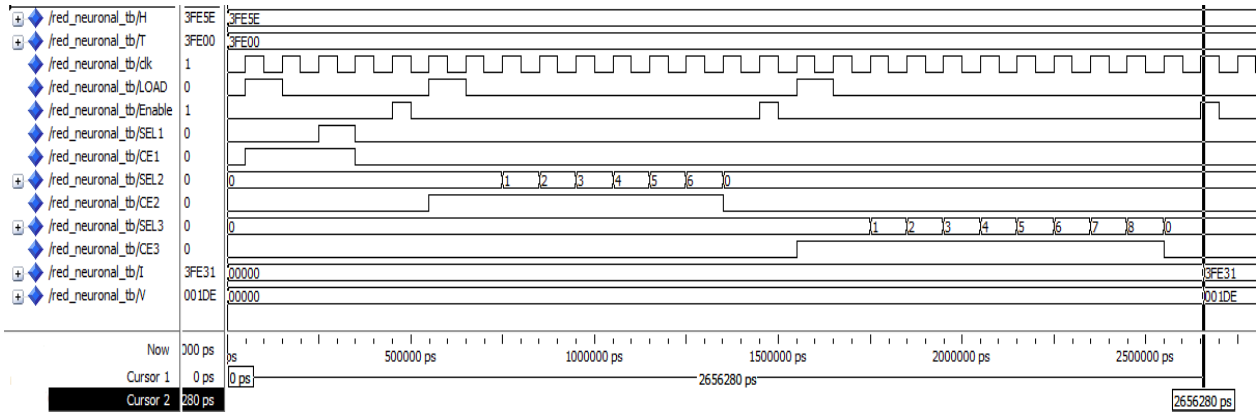


Fig. (4.8) *Respuesta de la red neuronal digital simple.*

Como se puede observar, la forma en la cual se trabajaron las señales de control hacen que el tiempo de respuesta de la red neuronal sea más lento, debido a que las neuronas de las siguientes capas no pueden comenzar a procesar información, hasta que las neuronas de la capa anterior hayan concluido. Este tiempo de respuesta se puede observar en la Fig. (4.8), donde se muestra la simulación temporizada que se le realizó a la red neuronal digital simple, ante un valor de entrada, obteniendo un tiempo de respuesta de la red de aproximadamente 2.6 milisegundos.

A continuación se muestra la simulación temporizada que se le realizó a la red neuronal digital simple, ante un valor de entrada, obteniendo un tiempo de respuesta de la red de 2656 nanosegundos, con las señales descritas anteriormente.

4.7 Red Neuronal Digital con técnica Pipeline.

Una vez que se analizó la respuesta de la red neuronal digital simple, se buscó optimizar el funcionamiento de la misma, a partir de la integración de 3 módulos denominados Controlador, Bloque de Entrada y Bloque de Salida. Se comenzará explicando el módulo Controlador y su funcionalidad al aplicar la técnica pipeline a la red neuronal. Fig. (4.9).

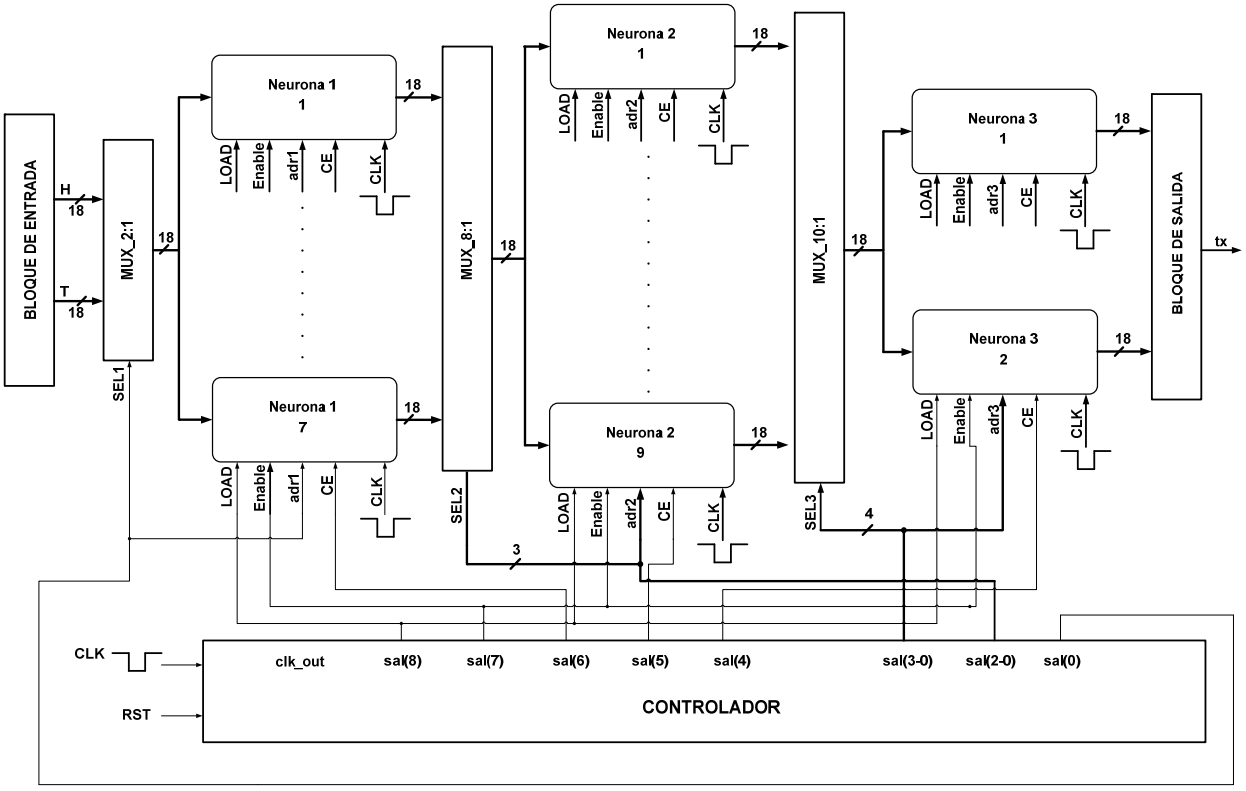


Fig. (4.9) *Diagrama de la Red neuronal Digital con técnica Pipeline.*

4.7.1 Técnica Pipeline.

La técnica Pipeline es un término perteneciente a la Ingeniería de Software, la cual consiste en hacer que una instrucción empiece a ejecutarse antes de que hayan terminado las anteriores, permitiendo de esta manera que varias instrucciones estén ejecutándose simultáneamente, por tal razón esta técnica permite reducir la longitud de los ciclos de procesamiento. Lo que se busca al

implementar esta técnica en nuestra red neuronal digital, es reducir el tiempo de respuesta de la red, para esto, basándose en la definición explicada anteriormente se logra que todas las neuronas contenidas en todas las capas trabajen al mismo tiempo. Para lograr esto, se creó el siguiente diagrama de tiempos el cual se explicará a continuación.

4.7.2 Diagrama de tiempo de las Señales de Control.

En la Fig. (4.10) se puede observar el diagrama de tiempo que se propuso para la optimización de la respuesta de la red neuronal; como se puede observar se emplea una señal denominada **sal**, la cual será de tipo vector y estará conformada por las señales de control que hemos estado manejando.

$sal(8) \rightarrow LOAD$
$sal(7) \rightarrow Enable$
$sal(6) \rightarrow CE1$
$sal(5) \rightarrow CE2$
$sal(4) \rightarrow CE3$
$sal(3-0) \rightarrow SEL3$
$sal(2-0) \rightarrow SEL2$
$sal(0) \rightarrow SEL1$

El principio de funcionamiento para las señales de control es el mismo que se explicó en la sección anterior, de tal manera que las diferencias que se tienen en este diagrama con el anterior son las siguientes:

- Para lograr que las 18 neuronas trabajen de forma simultánea, se activaron al mismo tiempo las señales CE1, CE2 y CE3, por tal motivo al estar operando simultáneamente solamente se necesitó únicamente un ciclo de operación de las señales LOAD y Enable, en vez de 3.
- En vez de tener de forma independiente a las señales SEL1, SEL2 y SEL3, se realizó la integración de estas señales en los 4 bits menos significativos de la señal sal; esto

ocasiona que exista una reducción importante en el número de señales entre un diagrama y otro.

- Como se puede ver en el diagrama de tiempo sin técnica pipeline, éste presenta 28 ciclos de operación de la señal de reloj clk con una frecuencia de 10 MHz; por otro lado, en el diagrama de tiempo con técnica pipeline se utilizaron únicamente aproximadamente 13 ciclos de operación para la misma señal con la misma frecuencia de operación clk_out.

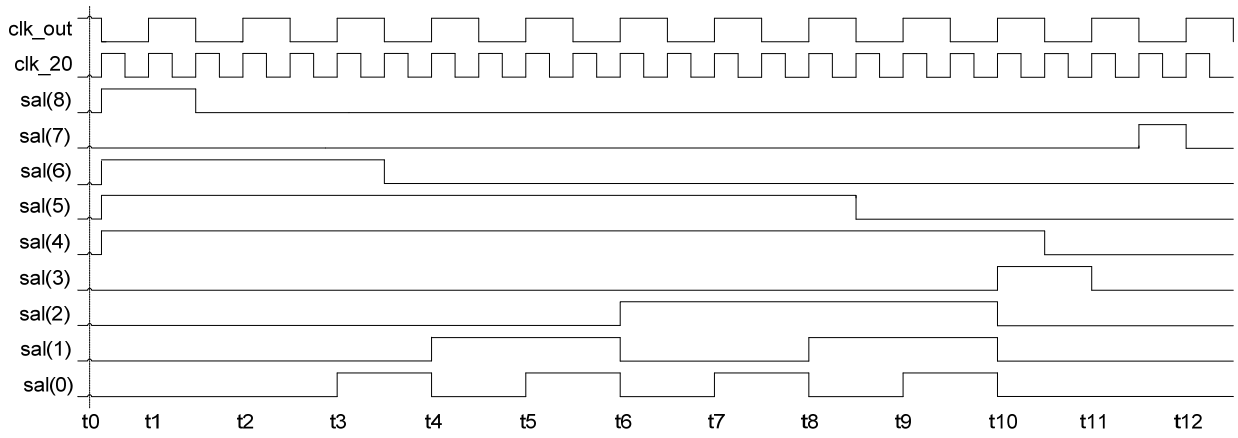


Fig. (4.10) *Diagrama de tiempo con técnica Pipeline.*

4.7.3 Módulo Controlador.

Para la generación de las señales de control se creó una entidad denominada **Controlador**, la cual se encuentra conformada por un módulo llamado PLL, que servirá para dividir la frecuencia de operación de la señal de reloj del FPGA (100 MHz), en dos frecuencias: **clk_out** la cual va a trabajar con una frecuencia de 10MHz y la señal **clk_20** que tendrá una frecuencia de 20MHz. Este módulo PLL, se creó utilizando la herramienta IP (CORE Generator & Architecture Wizard) de Xilinx. La elección de la frecuencia de operación de 10MHz, se debió principalmente para evitar valores transitorios durante la asignación de los valores a las señales y la señal de 20MHz se utilizará para el funcionamiento del controlador como se podrá ver a continuación.

Para reproducir el diagrama de tiempo de la Fig. (4.10) nos auxiliamos del flanco de subida de la señal de reloj clk_20 y entre cada flanco de subida se le asignó el valor deseado a cada una de las señales de control. Por ejemplo, en la Fig. (4.11) en el tiempo t1, se puede apreciar el valor que se asignó a cada una de las señales de control en ese instante de tiempo, esto mismo se realizó a lo largo de todo el diagrama hasta llegar al tiempo t24.

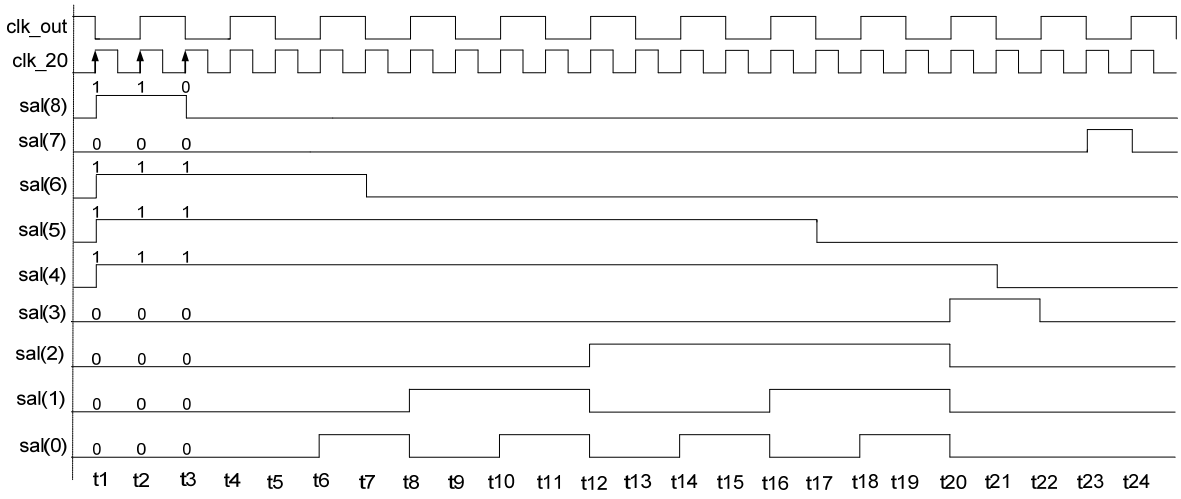


Fig. (4.11) *Análisis de las señales de control con técnica pipeline.*

Una vez que se obtuvieron los valores de todas las señales, se conformaron arreglos de bits para cada instante de tiempo. Por ejemplo, en el instante de tiempo t1 se tiene el siguiente arreglo de bits "101110000", el cual para facilitar su descripción en lenguaje VHDL se representó en formato hexadecimal de la siguiente manera '1' & X"70" donde el bit más significativo '1' se va a concatenar al valor hexadecimal "70", el cual representa el resto de nuestro arreglo de bits "01110000". En el siguiente listado se muestra parte del código descrito en VHDL para la entidad Controlador. Se puede observar que se construye a partir del uso de una estructura de tipo ***case*** y de un contador denominado **aux**, el cual se utilizará para hacer referencia a los tiempos descritos anteriormente. Como se puede ver en las líneas 32 y 33, este contador se incrementará cada vez que aparezca un flaco de subida en la señal de reloj clk_20, permitiendo de esta manera que en cada cuenta se le asigne el valor del arreglo conformado, al puerto de salida del controlador denominado como sal, líneas (34-38)**.** En la parte del listado que se muestra abajo, solamente se mostraron los tres primeros casos de la estructura antes mencionada; para consultar el código completo, ver el Apéndice D, Sección 5, Listado (1).

```
32          elsif (clk_20'event and clk_20 = '1') then
33          aux := aux + 1;

34          case aux is
36          when "000001" => sal <= '1' & X"70";
37          when "000010" => sal <= '1' & X"70";
38          when "000011" => sal <= '0' & X"70";
```

Listado (4.4) *Descripción del comportamiento de la entidad Controlador.*

4.7.3.1 Respuesta del módulo controlador.

Se puede apreciar en la Fig. (4.12a) la simulación temporizada del módulo controlador, con las formas de onda de las señales de control descritas con anterioridad. Se puede ver que se tiene un tiempo de respuesta del controlador de 1150 nanosegundos.

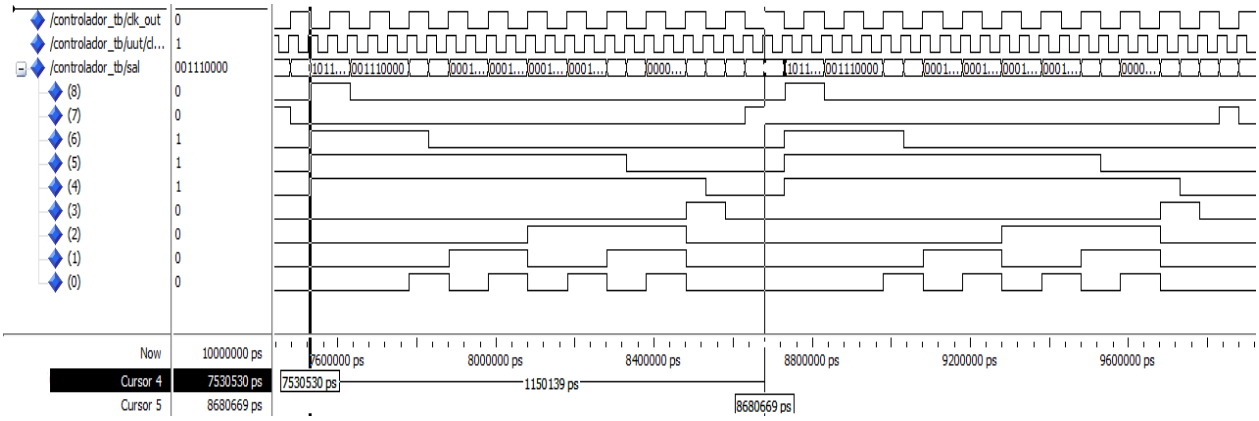


Fig. (4.12a) *Simulación del funcionamiento del Módulo Controlador.*

También se observó un tiempo de propagación entre cada cambio de valor de señal de aproximadamente 0.2 nanosegundos, Fig. (4.12b).

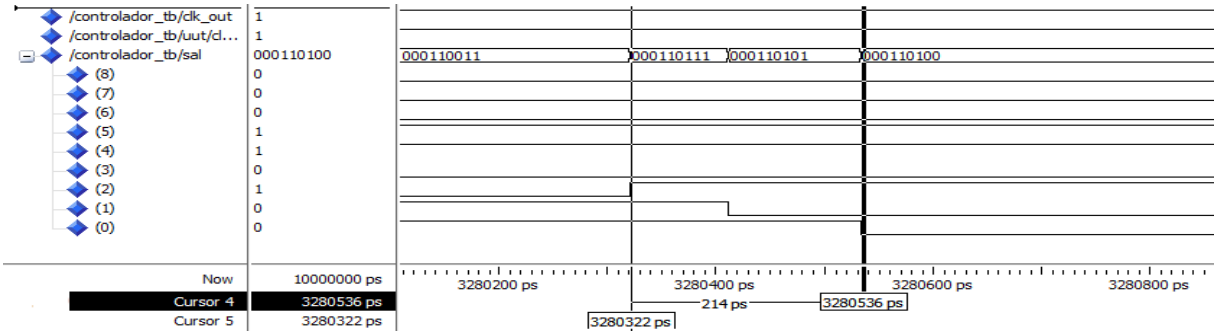


Fig. (4.12b)*Medición del tiempo de propagación del Módulo Controlador.*

4.8 Bloque de Entrada.

Otro módulo que se integró al diagrama de la red neuronal digital, fue el denominado Bloque de Entrada el cual sirvió para almacenar los 165 datos de Irradiancia y los 165 datos de Temperatura que se obtuvieron del subconjunto de datos de prueba. Estos datos se almacenaron en dos

memorias RAM (Memoria de Acceso Aleatorio) configuradas en el formato de tipo lectura una para cada variable. Ver Apéndice D, Sección 5, Listado (2) para la descripción en VHDL.

4.9 Bloque de salida.

Para poder hacer la extracción de los resultados obtenidos de la red neuronal implementada en el FPGA, para su análisis, se tuvo que elaborar un módulo denominado Bloque de Salida, el cual va a estar conformado en su interior por 2 memorias RAM, con configuración escritura/lectura cada una. Ya que cada vez que la red neuronal digital calcule un valor de corriente a corto circuito y un valor de voltaje a circuito abierto, estos valores se irán escribiendo en sus respectivas memorias, esto se repetirá hasta que la red neuronal digital haya terminado de procesar todos los datos contenidos en las memorias de entrada. Una vez que se han llenado ambas memorias de salida, se cambiará la configuración de las mismas a tipo lectura, para de esta manera, por el puerto de salida **tx**, ir sacando los resultados con el protocolo de comunicación serial.

A continuación, se muestra el diagrama con todos los módulos utilizados para el desarrollo del bloque de salida Fig. (4.13). La descripción de cada uno se puede apreciar en el Apéndice D, Sección 5, Listados (3-7).

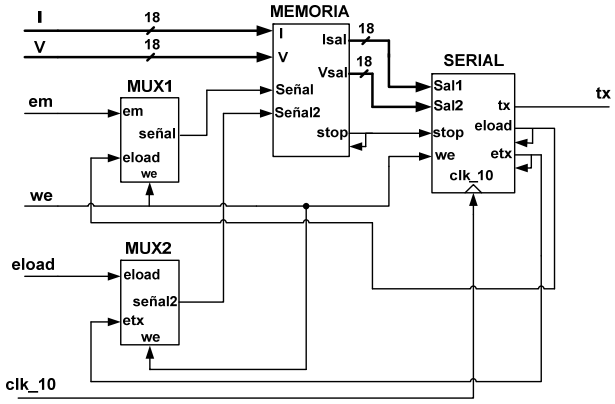


Fig. (4.13) *Módulos que integran al bloque de salida.*

Una vez que se desarrollaron los módulos mencionados anteriormente, se continuó con la integración de éstos a la entidad de mayor jerarquía denominada Red_Neurona_Pipeline; consultar el Apéndice D, Sección 5, Listado (8), para ver el código completo.

4.10 Respuesta de la Red Neuronal con técnica Pipeline.

Se realizó la simulación temporizada de la red neuronal digital con técnica pipeline, Fig. (4.14a), y se observó que la red neuronal presenta un tiempo de respuesta de aproximadamente de 1200 nanosegundo para cada valor de entrada.

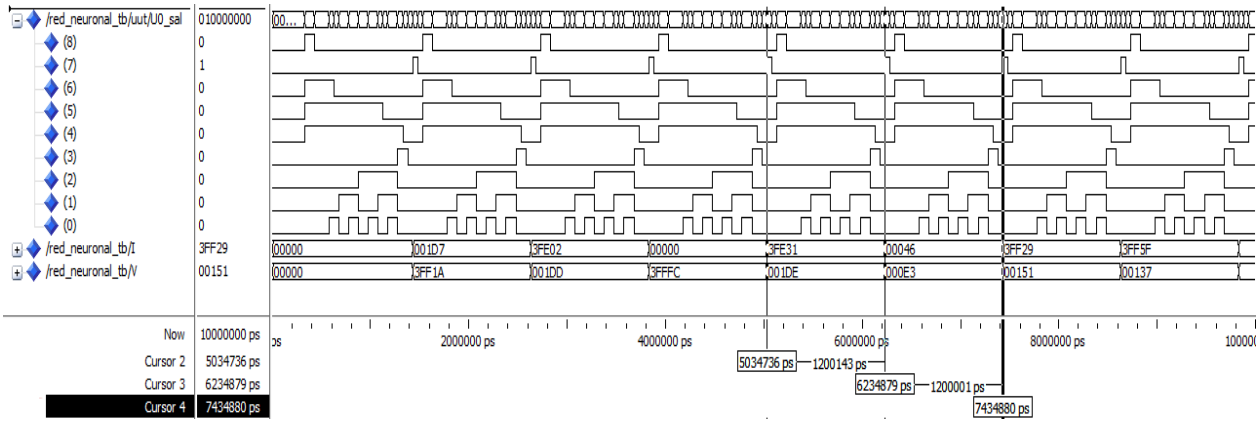


Fig. (4.14a) *Respuesta de la Red Neuronal Digital con técnica Pipeline.*

El tiempo en que la red neuronal procesó los 165 datos contenidos en los bloques de entrada, fue de aproximadamente 200,000 nanosegundos, ver Fig. (4.14b).

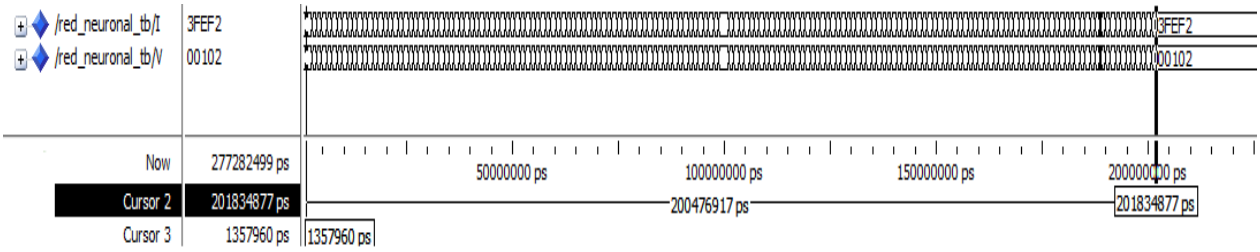


Fig. (4.14b) *Tiempo de respuesta de la red neuronal ante todos los datos de entrada.*

En la Fig. (4.15) se muestra un tiempo transitorio de aproximadamente 0.05 nanosegundos, entre cada valor de salida de la red neuronal digital.

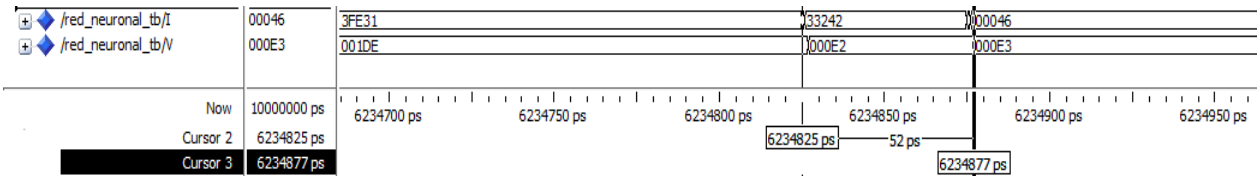


Fig. (4.15) *Transitorio de la Red Neuronal Digital con técnica Pipeline.*

En la siguiente Fig. (4.16a) se muestra a la unidad de procesamiento MAC, utilizada para todas las neuronas de las tres capas.

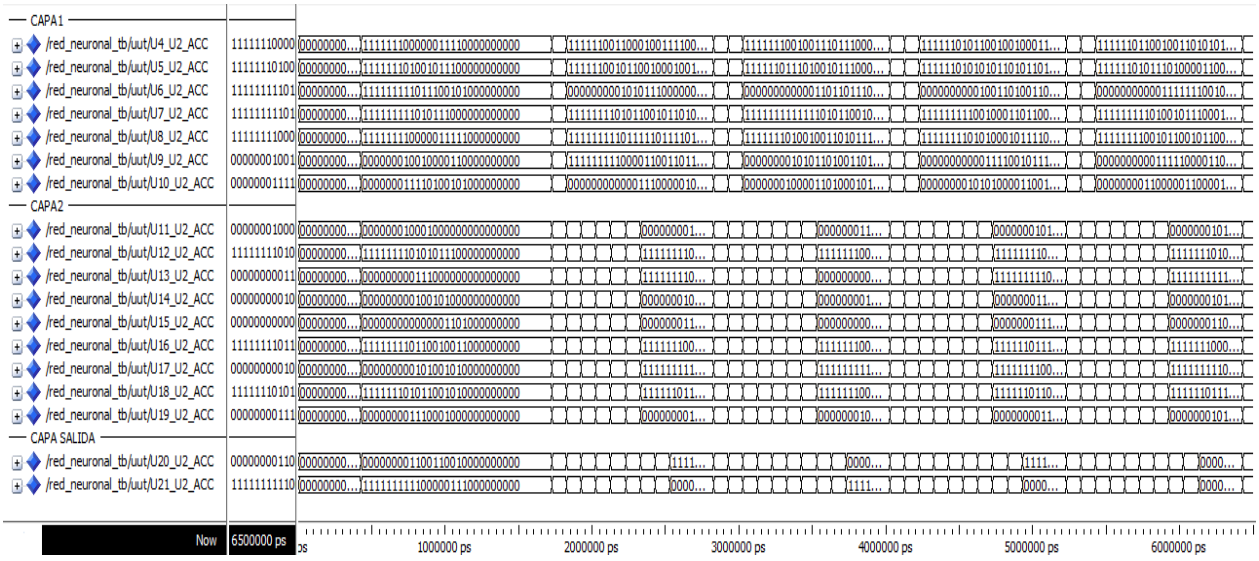


Fig. (4.16a) *Unidades de Procesamiento MAC para las neuronas de todas las capas.*

Se observó un tiempo de respuesta de 0.2 microsegundos para la unidad MAC, contenida en las neuronas que conforman a la capa 1, ver Fig. (4.16b).

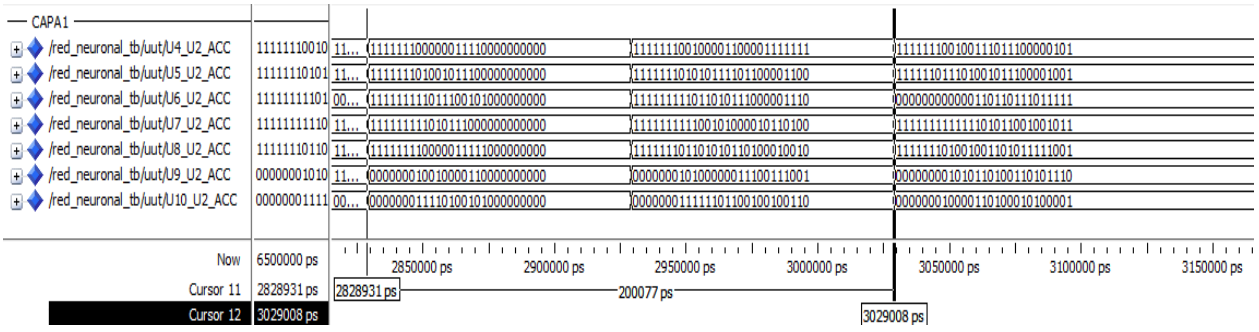


Fig. (4.16b) *Tiempo de respuesta de la unidad MAC para la capa 1.*

Para las 9 neuronas de la capa 2 se obtuvo un tiempo de aproximadamente 0.7 microsegundos, mostrado en la Fig. (4.16c).

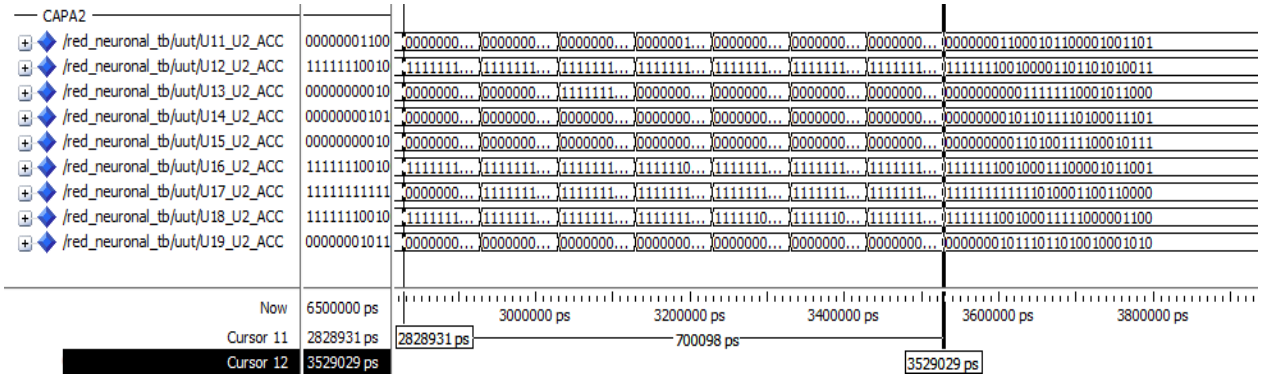


Fig. (4.16c) *Tiempo de respuesta de la unidad MAC para la capa 2.*

Y finalmente, para las 2 neuronas de la capa de salida, se obtuvo un tiempo de aproximadamente 0.9 microsegundos.

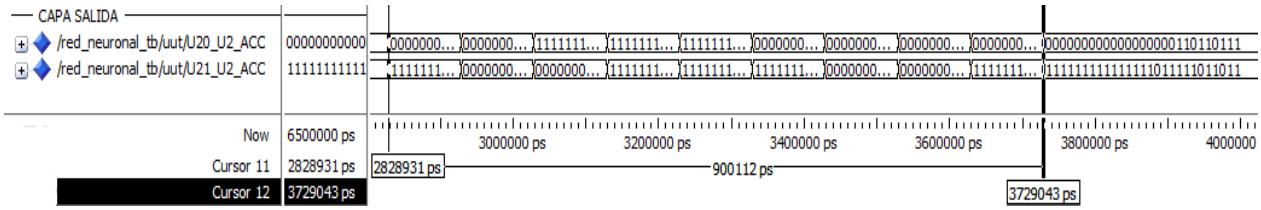


Fig. (4.16d) *Tiempo de respuesta de la unidad MAC para la capa de salida.*

A continuación, se muestra en la Fig. (4.17a) el comportamiento del Multiplexor 2:1 que se describió anteriormente, ante los valores de entrada (H,T) y su señal de salida **cm_x1** y con la señal de selección **sal(0)**.

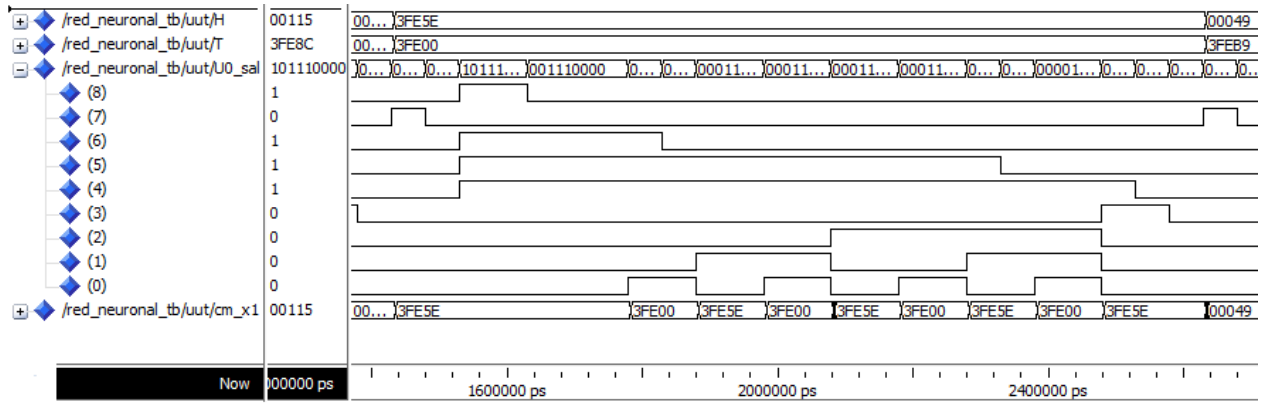


Fig. (4.17a) *Multiplexor 2:1.*

En esta Fig. (4.17b) se analiza el tiempo de respuesta del multiplexor, teniendo un tiempo de propagación de 1.1 nanosegundos para el primer dato y un tiempo de aproximadamente 4.3 nanosegundos para el segundo dato.

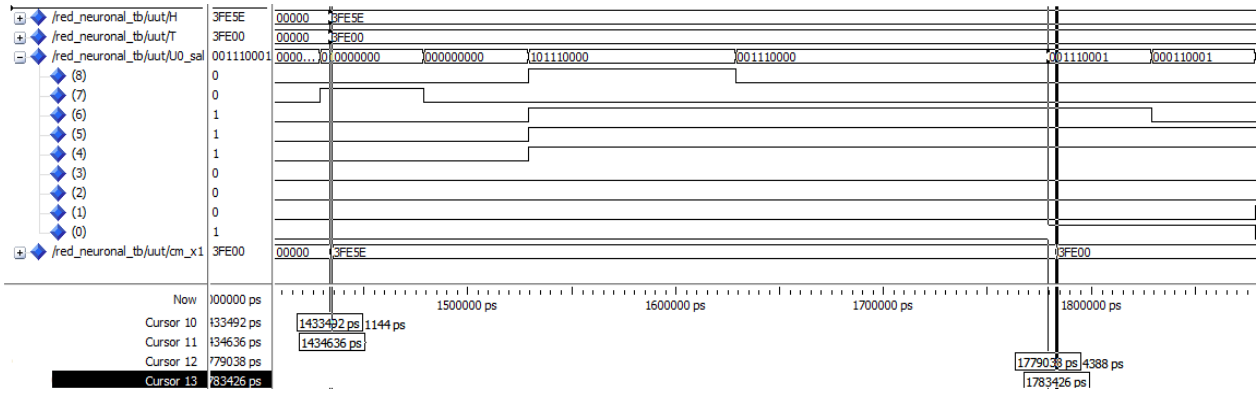


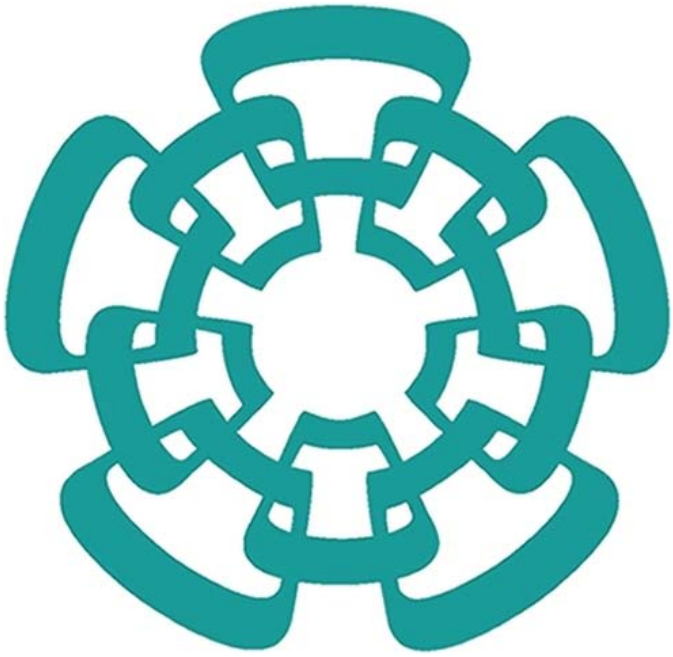
Fig. (4.17b) *Tiempo de respuesta del Multiplexor 2:1.*

Conclusiones.

En este capítulo se describió el procedimiento que se realizó para implementar la red neuronal artificial, propuesta en el Capítulo 3, en lenguaje VHDL. Se observó un tiempo de procesamiento menor de la red neuronal con la técnica pipeline, en comparación de esta misma red sin esta técnica, por lo cual se observó la efectividad de esta técnica. Se concluye que la implementación de este tipo de redes en un dispositivo FPGA es la más óptima debido al paralelismo con el que se puede trabajar. En el siguiente capítulo se discutirán de manera más general los resultados obtenidos en este tipo de dispositivos.

CAPÍTULO 5

Resultados generales



5.1 Introducción.

En este capítulo nos enfocaremos al análisis de los resultados obtenidos del dispositivo digital FPGA. Se realizará la comparación de estos datos con los datos del panel solar ST5 que fueron utilizados como salida objetivo. De igual manera, se obtendrán los resultados del Error Cuadrático Medio (MSE), el Coeficiente de Correlación de Pearson (R) y la Gráfica de Regresión, para de esta manera, poder concluir si el objetivo planteado al principio de este trabajo fue alcanzado.

Además, se mostrará con un analizador digital la respuesta del módulo controlador utilizado en el capítulo anterior.

5.2 Análisis de las señales del módulo controlador con técnica pipeline.

En la Fig. (5.1) se muestran las señales del controlador obtenidas para controlar el procesamiento de las 3 capas de la red neuronal. Esta medición se realizó utilizando el analizador digital MSO 2012 de la compañía Tektronix. En la figura (5.1) se pueden observar los 9 bits que conforman a la señal de salida del controlador denominado SAL.

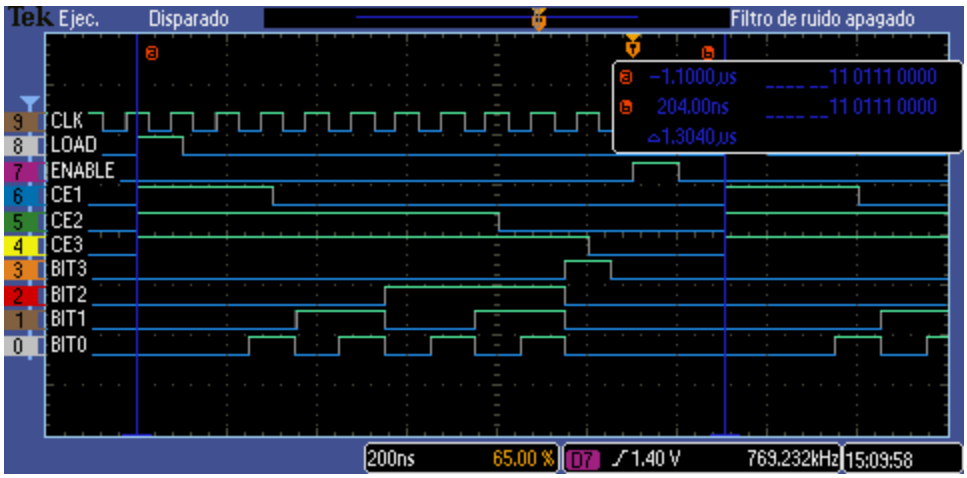


Fig. (5.1) *Señales de control con técnica Pipeline.*

Se pueden observar las tres señales que activan y desactivan a todas las unidades de procesamiento de la red neuronal digital (CE1, CE2, CE3), así como también se muestran las señales (bit 0 – bit3), las cuales como se mencionó en el capítulo anterior, se utilizarán para controlar el proceso de multiplicación y acumulación del módulo MAC. Como se puede

observar, se tiene un tiempo de respuesta del controlador de aproximadamente 1.3 microsegundos.

5.3 Extracción de los datos.

Una vez de que se implementó la red neuronal en el dispositivo FPGA, se realizó la transmisión de los resultados almacenados en las memorias de salida utilizando el protocolo de comunicación serial. Para esto, la tarjeta de desarrollo cuenta con un puerto USB-UART para permitir la comunicación entre la PC y la tarjeta a partir de un puerto COM; este puerto se puede observar en la Fig. (5.2).

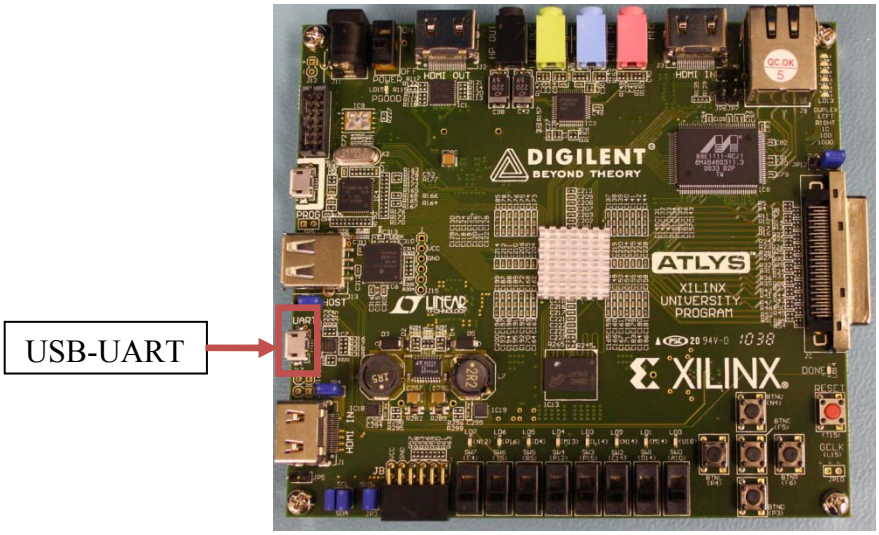


Fig. (5.2) *Puerto utilizado para la comunicación serial.*

Se utilizó el programa Advanced Serial Port Monitor Versión 4.3.7 de la compañía AGG Software, para la captura y la visualización de los datos en la PC. A continuación en la Fig. (5.3), se puede apreciar la interfaz del programa con los datos capturados, así como también se muestran algunas características que se usaron para la transmisión, como son la velocidad de transmisión de 9600 baudios, la cual se mencionó anteriormente, el puerto COM utilizado y el tamaño de la trama de los bits de datos enviados. Cabe mencionar que se tendrán 6 columnas, donde en las primeras 3 columnas se representa a un valor de corriente a corto circuito mientras que en las 3 últimas columnas se representa un valor de voltaje a circuito abierto. Debido a la velocidad de transmisión utilizada, el tiempo total en el que se transmitieron todos los datos fue de 1.1 segundos, el cual es un tiempo muy rápido.

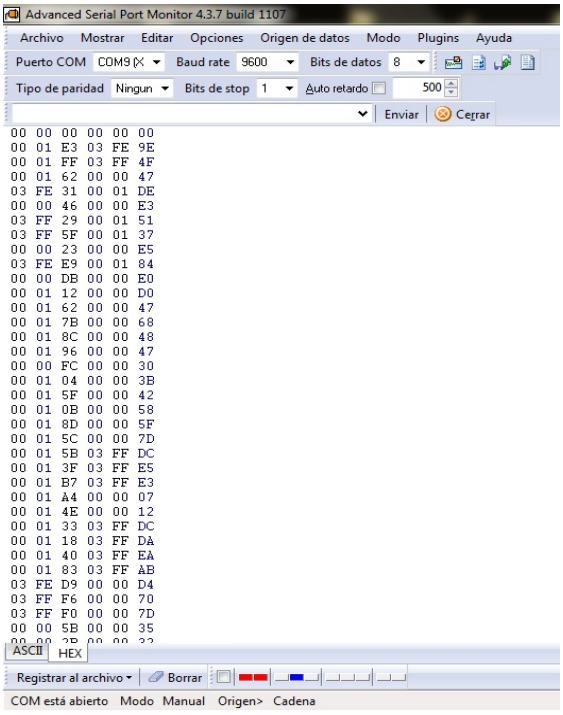


Fig. (5.3) *Interfaz utilizada del programa Advanced Serial Port Monitor.*

5.4 Comparación entre los datos del FPGA vs Modelo análogo eléctrico del panel solar ST5.

Una vez que se transmitieron todo los datos del FPGA y se capturaron en el programa descrito anteriormente, se realizaron dos gráficas comparativas para observar si la arquitectura de la red neuronal artificial descrita en el Capítulo 3 es un buen aproximador de funciones. A continuación en la Fig. (5.4a), se muestra en color rojo, el perfil de la corriente a corto circuito que se obtuvo del modelo del panel ST5 al que se quiere aproximar y en color azul se muestra la aproximación que se obtuvo con el dispositivo FPGA. Como se puede apreciar, a simple vista existe una buena aproximación entre estas dos gráficas; también se realizó el cálculo del Error Cuadrático Medio obteniendo un valor de 2.0182E-07, el cual es un resultado muy bueno. De igual manera se graficaron los datos para el voltaje a circuito abierto, Fig. (5.4b), obteniendo de esta manera un valor de MSE = 10E-04.

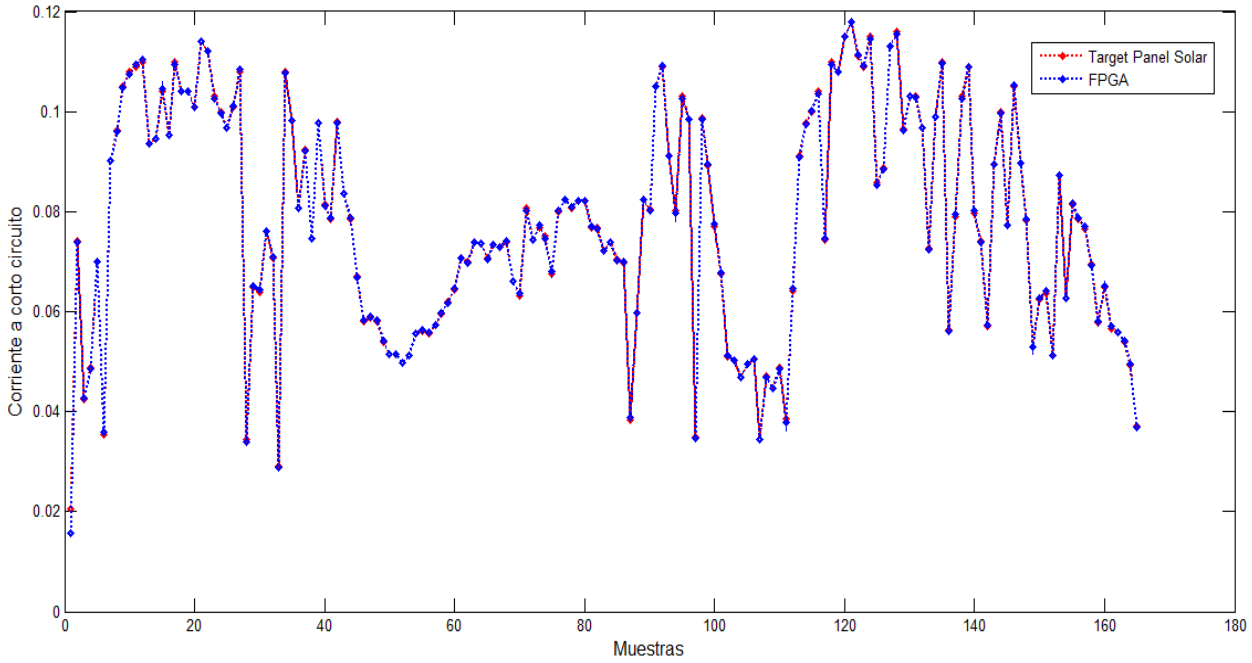


Fig. (5.4a) *Gráfica de comparación entre los datos de corriente a corto circuito del FPGA y los valores target del modelo del panel solar ST5.*

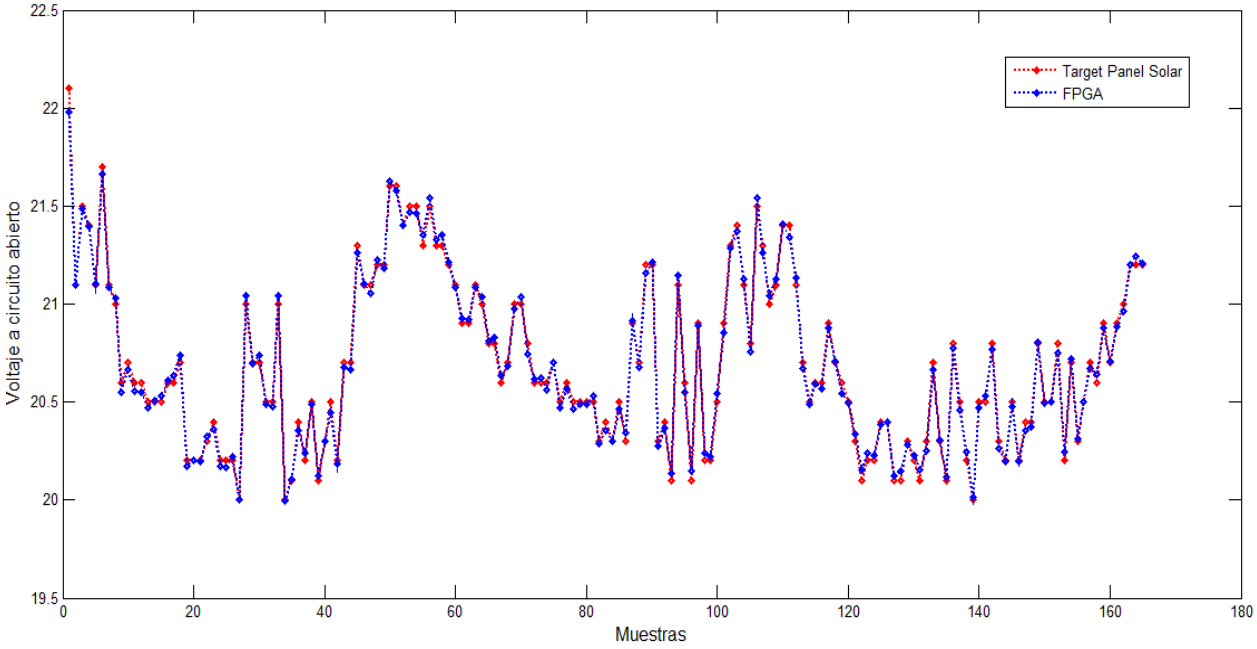


Fig. (5.4b) *Gráfica de comparación entre los datos de voltaje a circuito abierto del FPGA y los valores target del modelo del panel solar ST5.*

5.5 Gráfica de Regresión y Coeficiente de Correlación.

Como se mencionó en la introducción de este capítulo, otro análisis que se realizó fue el del coeficiente de correlación de Pearson (R) obteniendo un valor de 0.9998 para los datos de la corriente a corto circuito, el cual por ser un valor muy cercano a la unidad, es un claro indicativo de que existe una buena dependencia entre los datos obtenidos del FPGA y los datos utilizados como salida objetivo. Además, se elaboró la gráfica de dispersión, con su respectiva ecuación característica en la Fig. (5.5a). Esta misma gráfica se realizó para los datos del voltaje a circuito abierto, Fig. (5.5b), obteniendo un coeficiente de correlación R = 0.9972 cercano a la unidad; de igual manera, en esta figura se puede observar la línea de regresión y su ecuación característica.

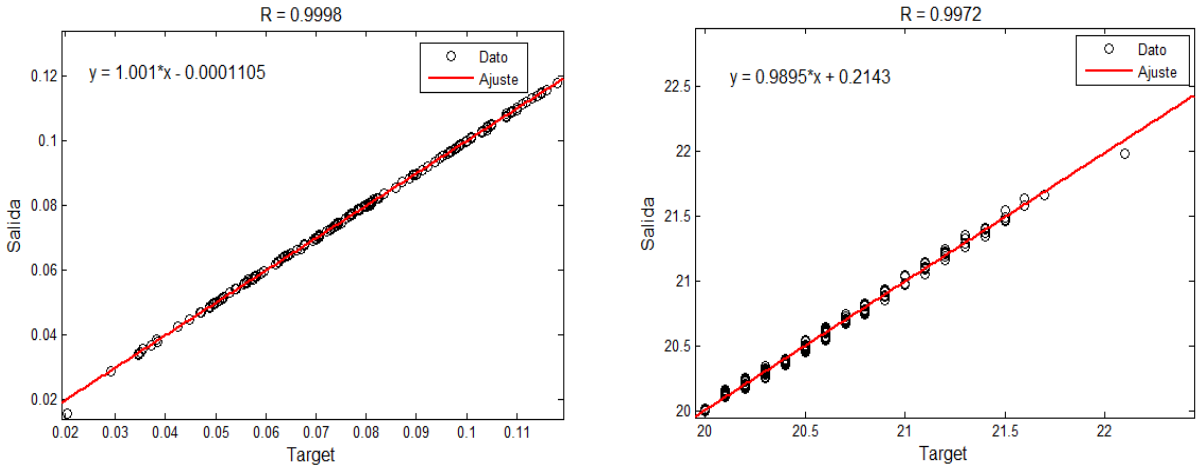
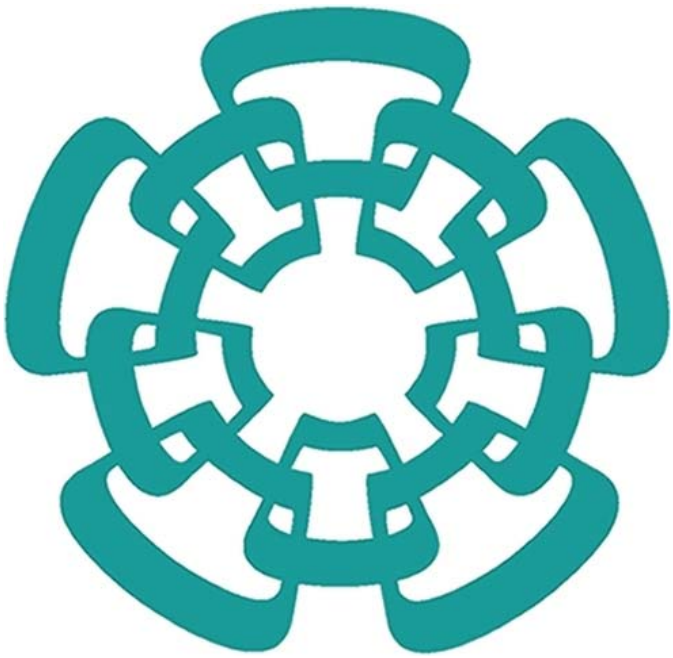


Fig. (5.5) *(a) Gráfica de Regresión de la Corriente a corto circuito. (b) Gráfica de Regresión del voltaje a circuito abierto.*

CAPÍTULO 6

Conclusiones



Conclusiones.

En la introducción de este trabajo se señaló que nuestro objetivo principal era la creación de un emulador de un panel fotovoltaico que pudiera imitar la respuesta eléctrica de un módulo solar comercial; este objetivo como se observó anteriormente fue alcanzado.

Se comprobó que las redes neuronales artificiales son una gran herramienta para solucionar problemas donde es difícil la obtención de un modelo matemático, como es el caso de problemas donde influyen parámetros ambientales, como es nuestro caso, ofreciendo de esta manera una nueva alternativa para el modelado de sistemas fotovoltaicos.

El emulador digital que se propuso en este trabajo, permitió tener una respuesta muy cercana al modelo análogo – eléctrico, que se propone en la referencia [1], lo cual nos indica que el uso de redes neuronales implementadas en dispositivos digitales, son otra alternativa para el estudio de los sistemas fotovoltaicos, ofreciendo la característica que puede ser en tiempo real.

La implementación de una red neuronal en un dispositivo digital, como se comprobó, puede brindarnos información sobre el comportamiento que tendría un módulo solar de cualquier dimensión, generando de esta manera un gran beneficio en el costo, ya que la adquisición de un emulador digital resulta más económica que un módulo solar.

Como se pudo observar, a lo largo de este trabajo se utilizaron varias herramientas para poder alcanzar nuestro objetivo, por tal motivo el desarrollo de este trabajo permitió consolidar el conocimiento en dichas herramientas de diseño profesional.

Optimización.

A fin de hacer más eficiente nuestra red neuronal artificial, se pueden realizar ciertas mejoras como son: realizar nuevamente el entrenamiento de la red neuronal, hasta obtener un valor de error MSE muy cercano a cero y un coeficiente de correlación R muy próximo a la unidad. Se puede además incrementar la base de datos meteorológica a un número mayor de años, para reducir estos parámetros.

Otra mejora sería aumentar el número de bits (18 bits), con el que se trabajó, para de esta manera tener una mayor resolución y reducir el error MSE y el valor del coeficiente de correlación

obtenidos por el FPGA, ya que al realizar este cambio la respuesta el FPGA se aproximará más a la respuesta de un módulo solar comercial.

Se pueden realizar mejoras en los códigos utilizados para la creación de la red neuronal digital, así como buscar reducir el número de rectas utilizadas para aproximar a la función de activación tansigmoidal.

También, se puede buscar la manera de reducir el número de multiplicadores DSP´s utilizados, ya que como se pudo observar, se utilizó un número considerable del número total que integran al FPGA Spartan-6 LX45, modelo CSG324C.

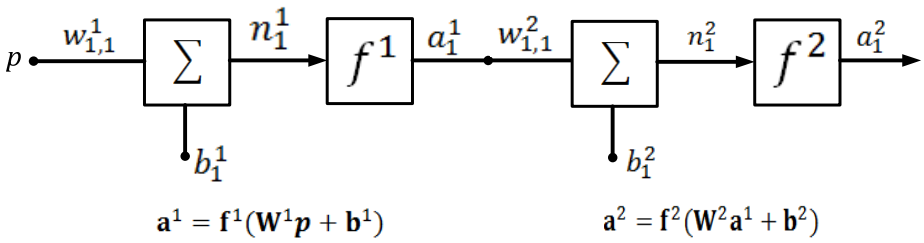
Trabajo Futuro.

Se han discutido en la sección anterior algunas optimizaciones que podrían darse como trabajo futuro, además de poder buscar emular otro tipo de módulos solares que existen en el mercado. También como trabajo futuro se puede buscar añadir sensores de medición de la irradiancia y la temperatura, para de esta manera obtener valores de corriente a corto circuito y voltaje a circuito abierto en tiempo real.

También se puede buscar la manera realizar la integración de una batería, un regulador y un inversor para de esta manera tener un sistema fotovoltaico completo en el FPGA. Es claro que las posibles optimizaciones son numerosas ofreciendo la oportunidad de, en un futuro continuar, investigando.

Apéndice A.

Sección 1. Cálculo de la matriz Jacobiana



Las funciones de transferencias utilizadas para cada neurona son las siguientes:

Para la neurona 1

$$f^1(n) = (n)^2$$

Para la neurona 2

$$f^2(n) = n$$

Por lo cual, las correspondientes derivadas quedarán expresadas como sigue:

$$f^1(n) = 2n\,, \quad f^2(n) = 1$$

El conjunto de entrenamiento se definirá como:

$$\{(p_1 = [1]), (t_1 = [1])\}\ ,\{(p_2 = [2]), (t_2 = [2])\}$$

Y la inicialización de los pesos y polarizaciones será la siguiente:

$$w^1 = [1]\,, b^1 = [0]\,, w^2 = [2]\,, b^2 = [1]$$

El primer paso es propagar las entradas a través de la red y calcular los errores.

$$a_1^0 = p_1 = [1]$$

$$n_1^1 = w^1 a_1^0 + b^1 = [1][1] + [0] = [1]$$

$$a_1^1 = f^1(n_1^1) = ([1])^2 = [1]$$

$$n_1^2 = w^2 a_1^1 + b^2 = [2][1] + [1] = [3]$$

$$a_1^2 = f^2(n_1^2) = ([3]) = [3]$$

$$e_1 = (t_1 - a_1^2) = ([1] - [3]) = [-2]$$

$$a_2^0 = p_2 = [2]$$

$$n_2^1 = w^1 a_2^0 + b^1 = [1][2] + [0] = [2]$$

$$a_2^1 = f^1(n_2^1) = ([2])^2 = [4]$$

$$n_2^2 = w^2 a_2^1 + b^2 = [2][4] + [1] = [9]$$

$$a_2^2 = f^2(n_2^2) = ([9]) = [9]$$

$$e_1 = (t_1 - a_1^2) = ([2] - [9]) = [-7]$$

El siguiente paso es inicializar y propagar hacia atrás la sensibilidad Marquardt usando las ecuaciones (1.10) y (1.11), para esto se calcula cada elemento de la matriz.

$$\mathbf{S}_1^{\sim 2} = -\mathbf{F}^2(\mathbf{n}_1^2) = -[1]$$

$$\mathbf{S}_1^{\sim 1} = \mathbf{F}^1(\mathbf{n}_1^1)(\mathbf{W}^2)^T \mathbf{S}_1^{\sim 2} = \left[2n_{1,1}^1\right][2][-1] = [2(1)][2][-1] = [-4]$$

$$\mathbf{S}_2^{\sim 2} = -\mathbf{F}^2(\mathbf{n}_2^2) = -[1]$$

$$\mathbf{S}_2^{\sim 1} = \mathbf{F}^1(\mathbf{n}_2^1)(\mathbf{W}^2)^T \mathbf{S}_2^{\sim 2} = \left[2n_{1,2}^2\right][2][-1] = [2(2)][2][-1] = [-8]$$

Se forma la matriz de Marquardt con los elementos calculados anteriormente usando la ecuación (1.12).

$$\mathbf{S}^{\sim 1} = [\mathbf{S}_1^{\sim 1} | \mathbf{S}_2^{\sim 1}] = [-4 \ -8]$$

$$\mathbf{S}^{\sim 2} = [\mathbf{S}_1^{\sim 2} | \mathbf{S}_2^{\sim 2}] = [-1 \ -1]$$

Se crea la matriz Jacobiana con la expresión (1.6) y se calculan sus componentes con las ecuaciones (1.7) y (1.8).

$$\mathbf{J(x)} = \begin{bmatrix} \frac{\partial e_{1,1}}{\partial w_{1,1}^1} & \frac{\partial e_{1,1}}{\partial b_1^1} & \frac{\partial e_{1,1}}{\partial w_{1,1}^2} & \frac{\partial e_{1,1}}{\partial b_1^2} \\ \frac{\partial e_{1,2}}{\partial w_{1,1}^1} & \frac{\partial e_{1,2}}{\partial b_1^1} & \frac{\partial e_{1,2}}{\partial w_{1,1}^2} & \frac{\partial e_{1,2}}{\partial b_1^2} \end{bmatrix}$$

$$[\mathbf{J}]_{1,1} = \frac{\partial e_{1,1}}{\partial w_{1,1}^1} = \frac{\partial e_{1,1}}{\partial n_{1,1}^1} x \frac{\partial n_{1,1}^1}{\partial w_{1,1}^1} = s_{1,1}^{\sim 1} x \frac{\partial n_{1,1}^1}{\partial w_{1,1}^1} = s_{1,1}^{\sim 1} x a_{1,1}^0 = (-4)(1) = -4$$

$$[\mathbf{J}]_{1,2} = \frac{\partial e_{1,1}}{\partial b_1^1} = \frac{\partial e_{1,1}}{\partial n_{1,1}^1} x \frac{\partial n_{1,1}^1}{\partial b_1^1} = s_{1,1}^{\sim 1} x \frac{\partial n_{1,1}^1}{\partial b_1^1} = s_{1,1}^{\sim 1} = -4$$

$$[\mathbf{J}]_{1,3} = \frac{\partial e_{1,1}}{\partial w_{1,1}^2} = \frac{\partial e_{1,1}}{\partial n_{1,1}^2} x \frac{\partial n_{1,1}^2}{\partial w_{1,1}^2} = s_{1,1}^{\sim 2} x \frac{\partial n_{1,1}^2}{\partial w_{1,1}^2} = s_{1,1}^{\sim 2} x a_{1,1}^1 = (-1)(1) = -1$$

$$[\mathbf{J}]_{1,4} = \frac{\partial e_{1,1}}{\partial b_1^2} = \frac{\partial e_{1,1}}{\partial n_{1,1}^2} x \frac{\partial n_{1,1}^2}{\partial b_1^2} = s_{1,1}^{\sim 2} x \frac{\partial n_{1,1}^2}{\partial b_1^2} = s_{1,1}^{\sim 2} = -1$$

$$[\mathbf{J}]_{2,1} = \frac{\partial e_{1,2}}{\partial w_{1,1}^1} = \frac{\partial e_{1,2}}{\partial n_{1,2}^1} x \frac{\partial n_{1,2}^1}{\partial w_{1,1}^1} = s_{1,2}^{\sim 1} x \frac{\partial n_{1,2}^1}{\partial w_{1,1}^1} = s_{1,2}^{\sim 1} x a_{1,2}^0 = (-8)(2) = -16$$

$$[\mathbf{J}]_{2,2} = \frac{\partial e_{1,2}}{\partial b_1^1} = \frac{\partial e_{1,2}}{\partial n_{1,2}^1} x \frac{\partial n_{1,2}^1}{\partial b_1^1} = s_{1,2}^{\sim 1} x \frac{\partial n_{1,2}^1}{\partial b_1^1} = s_{1,2}^{\sim 1} = -8$$

$$[\mathbf{J}]_{2,3} = \frac{\partial e_{1,2}}{\partial w_{1,1}^2} = \frac{\partial e_{1,2}}{\partial n_{1,2}^2} x \frac{\partial n_{1,2}^2}{\partial w_{1,1}^2} = s_{1,2}^{\sim 2} x \frac{\partial n_{1,2}^2}{\partial w_{1,1}^2} = s_{1,2}^{\sim 2} x a_{1,2}^1 = (-1)(4) = -4$$

$$[\mathbf{J}]_{2,4} = \frac{\partial e_{1,2}}{\partial b_1^2} = \frac{\partial e_{1,2}}{\partial n_{1,2}^2} x \frac{\partial n_{1,2}^2}{\partial b_1^2} = s_{1,2}^{\sim 2} x \frac{\partial n_{1,2}^2}{\partial b_1^2} = s_{1,2}^{\sim 2} = -1$$

Entonces la matriz Jacobiana queda de la siguiente manera:

$$\mathbf{J(x)} = \begin{bmatrix} -4 & -4 & -1 & -1 \\ -16 & -8 & -4 & -1 \end{bmatrix}$$

Apéndice B.

Sección 1. Hoja de datos del Panel Solar ST5.

SIEMENS

Solar module ST5

The Siemens ST5 solar (photovoltaic) module converts energy contained in sunlight directly into electricity. It is a solid-state generator that operates silently, without fuel, waste or pollution.

Designed for use in 12 volt systems, the ST5 performs efficiently in all operational conditions. Its distinctive ability to deliver battery-charge power levels in low-light situations makes it particularly effective for specialized applications and in adverse or changeable environments.

Siemens advanced PowerMax® thin film technology

The ST5 module is composed of a monolithic structure of series-connected Copper Indium Diselenide (CIS) based solar cells. These multiple-layer cells, a product of Siemens proprietary material, structure and process technologies, called PowerMax® thin film technology, are characterized by exceptional spectral response and long-term performance integrity. They give the ST5 performance efficiency similar to crystalline photo-voltaic modules.

Engineered for durability and ease of use, the ST5 is manufactured under strict quality controls in ISO 9001 certified facilities. A black-anodized aluminum frame secures the glass-front laminate which encapsulates, protects and electrically isolates the solar cells. This rugged construction enables the ST5 to endure even severe environ-mental events and continue to generate power reliably and efficiently.

Exacting standards of quality plus the proven performance of Siemens advanced photovoltaic technologies, make the ST5 an ideal choice for select commercial and industrial solar systems.

Solar module	
Model:	**ST5**
Rated Power:	**5 Watts**
Limited Warranty:	**10 Years**

Advanced solar technology

- Proprietary multiple-layer CIS solar cells capture a broad spectrum of light energy to deliver excellent power performance – even in reduced-light or in poor weather conditions.
- A conductive front layer of zinc oxide provides superior light transmission and trapping properties to enhance power output.
- Monolithic interconnected structure of series-connected cells contributes to high reliability.

Durable construction

- Every module is subject to final factory review, inspection and test to assure compliance with electrical, mechanical and visual criteria.
- Ultra-clear tempered glass front provides excellent light transmission and protects from wind, hail, particle damage and impact.
- Solar cells are laminated between a multi-layered polymer backsheet and layers of ethylene vinyl acetate (EVA) for environmental protection, moisture resistance and electrical isolation.
- Torsion and corrosion resistant anodized-aluminum module frame helps assure dependable performance, even through harsh weather conditions.
- Durable multiple-layered backing system provides the module underside with protection from scratching, cuts, breakage and most environmental conditions.
- Laboratory tested for a wide range of operating conditions.
- Manufactured in ISO 9001 certified facilities to exacting Siemens quality standards.

Easy installation and use

- Two-conductor, 5' [1.54m], (minimum length), UV stable cable facilitates a variety of mounting schemes and permits easy field wiring.
- Pre-drilled frame mounting holes are strategically positioned for secure and easy installations.
- Uniform flat-black appearance of module and frame is visually compatible with a variety of installation requirements.

Performance warranty

- 10 Year limited warranty on power output.

Solar module ST5

Electrical parameters		
Maximum power rating P_{max}	[Wp] 1)	5
Rated current I_{MPP}	[A]	0.32
Rated voltage V_{MPP}	[V]	15.6
Short circuit current I_{SC}	[A]	0.37
Open circuit voltage V_{OC}	[V]	21.0

Thermal parameters		
NOCT 2)	[°C]	47±2
Temp. coefficient: short-circuit current		0.13mA / °C
Temp. coefficient: open-circuit voltage		-0.1V/ °C

Qualification test parameters			
Temperature cycling range		[°C]	-40 to +85
Maximum system voltage		[V]	25.0
Wind Loading	PSF	[N/m²]	50 [2400]
Maximum distortion 3)		[°]	1.2
Hailstone impact	Inches	[mm]	1.0 [25]
	MPH	[m/s]	52 [v=23]
Weight	Pounds	[kg]	3.0 [1.4]

1) Wp (Watt peak) = Peak power
(Minimum Wp = 4.5 Watts)

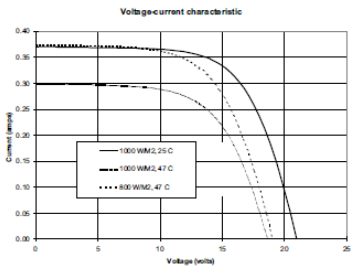
Under standard test conditions:

Air Mass	AM = 1.5
Irradiance	E = 1000 W/m²
Cell temperature	T_C = 25 °C

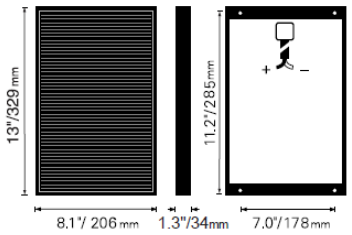
2) Normal Operating Cell Temperature at:

Irradiance	E = 800 W/m²
Ambient temperature	T_a = 20 °C
Wind Speed	v_W = 1 m/s

3) Diagonal lifting of module plane.



Module dimensions



Hole Diameter 0.26 inch (6.6mm)
Mounting hole dimensions are center to center

Your address for photovoltaics from Siemens Solar

Further information on solar products, systems, principles and applications is available in the Siemens Solar product catalog.

Siemens modules are recyclable.

Status 05/00 - Subject to modification.

Siemens Solar GmbH
A joint venture of
Siemens AG and Bayernwerk AG
Postfach 46 07 05
D-80915 München
Germany
Tel: 49-89-636-59001
Fax: 49-89-636-59434

Siemens Solar Industries
P.O.Box 6032
Camarillo, CA 93011, U.S.A.
Tel: 805-482-6800
Fax: 805-388-6395
Web site: www.siemenssolar.com
E-mail: ssi.sales@solar.siemens.com

Siemens Showa Solar Pte. Ltd.
166 Kallang Way
Singapore 349248
Tel: 65-842-3886
Fax: 65-842-3887

ST5 05/00

Sección 2. Código del Panel Solar ST5.

```
*MODULE_BEH.LIB

*   BEHAVIOURAL MODEL OF A PV MODULE
*   INPUT PARAMETERS: AREA, AM1.5 JSCMR, AM1.5 VOCMR, AM1.5, PMAXMR
*   AM1.5 VMMR, AM1.5 IMMR, CURRENT TEMP COEFF, VOLTAGE TEMP.COEFF,
*   NOCT, REFERENCE TEMPERATURE

*   NODES
*   (400) REFERENCE
*   (401) INTERNAL NODE
*   (402) INPUT IRRADIANCE
*   (403) INPUT, AMBIENT TEMPERATURE
*   (404) OUTPUT
*   (405) OUTPUT, (VOLTAGE) VALUE=SHORT CIRCUIT CURRENT(A) AT IRRADIANCE
*   AND TEMPERATURE
*   (406) OUTPUT, OPEN CIRCUIT VOLTAGE AT IRRADIANCE AND TEMPERATURE
*   (407) OUTPUT, VOLTAGE VALUE=CELL OPERATING TEMPERATURE(°C)
*   (408) OUTPUT, MPP CURRENT
*   (409) OUTPUT, MPP VOLTAGE

.subckt module_beh 400 402 403 404 405 406 407 408 409 params:
+iscmr=1, coef_iscm=1, vocmr=1, coef_vocm=1, pmaxmr=1,
+noct=1, immr=1, vmmr=1, tr=1, ns=1, np=1

girrad 400 401 value = {v(402)/1000*(iscmr+coef_iscm*(v(407)-25))}
eisc 405 400 value = {v(402)/1000*(iscmr+coef_iscm*(v(407)-25))}
evoc 406 400 value = {if(v(405)>1e-11, vocmr+coef_vocm*(v(407)-25)+8.66e-5*
+(v(407)+273)*log(v(405)/(iscmr)),0)}
etcell 407 400 value = {v(403)+(noct-20)/800*v(402)}

gidiode 401 400 value = {v(405)/(exp(v(406)/(ns*8.66e-5*(v(407)+273)))-1)*
+(exp(v(401)/(ns*8.66e-5*(v(407)+273)))-1)}
rsm 401 404 {vocmr/(iscmr)-pmaxmr/(iscmr**2*(vocmr/(ns*0.0258)-log
+((vocmr/(ns*0.0258))+0.72))/(1+vocmr/(ns*0.0258)))}
.func frsm() {vocmr/(iscmr)-pmaxmr/(iscmr**2*(vocmr/(ns*0.0258)-log
+((vocmr/(ns*0.0258))+0.72))/(1+vocmr/(ns*0.0258)))}

gim 400 408 value={immr*v(402)/1000+coef_iscm*(v(403)-25)}
rimm 408 400 1
evmm 409 400 value={if (v(402)>0.001, ns*8.66e-5*(v(407)+273)*log(1+(v(405)-
v(408))/v(405)*(exp(v(406)/
+(ns*8.66e-5*(v(407)+273)))-1))-v(408)*frsm,0)}
.ends module_beh
```

Apéndice C.

Sección 1. Código de la Red Neuronal en Matlab.

```
%Red Neuronal
load datos.dat;

%Creación y definicion de las entradas y capas de la red
net = network;
net.numInputs = 1;
net.numLayers = 3;

%Conexión de los bias con las capas
net.biasConnect(1) = 1;
net.biasConnect(2) = 1;
net.biasConnect(3) = 1;

%Conexión de las entradas
net.inputConnect(1,1) = 1;

%Conexión de las capas
net.layerConnect(2,1) = 1;
net.layerConnect(3,2) = 1;

%Conexion de la capa a la salida
net.outputConnect = [0 0 1];

%Input
net.inputs{1}.size = 2;
net.inputs{1}.processFcns = {'removeconstantrows','mapminmax'};
net.outputs{3}.processFcns = {'removeconstantrows','mapminmax'};

%Parametros de las capas de la red
net.layers{1}.size = 7;
net.layers{1}.transferFcn = 'tansig';
net.layers{1}.initFcn = 'initnw';

net.layers{2}.size = 9;
net.layers{2}.transferFcn = 'tansig';
net.layers{2}.initFcn = 'initnw';

net.layers{3}.size = 2;
net.layers{3}.transferFcn = 'tansig';
net.layers{3}.initFcn = 'initnw';

%Parametros del entrenamientos e inicializacion
net.initFcn = 'initlay';
net.performFcn = 'mse';
net.trainFcn = 'trainlm';
net.plotFcns = {'plotperform','plottrainstate','plotregression'};
net.divideFcn = 'dividerand';

%Inicialización
net = init(net);

%Datos de entrada
r = datos(:,1);
Radiacion = r';
t = datos(:,2);
Temperatura = t';

%Datos Target
```

```
59      i = datos(:,3);
60      Corriente = i';
61      v = datos(:,4);
62      Voltaje = v';
63
64      %Normalización de las entradas y los targets
65      [Norm_R,Param_R] = mapminmax(Radiacion,-1,1);
66      [Norm_T,Param_T] = mapminmax(Temperatura,-1,1);
67      [Norm_C,Param_C] = mapminmax(Corriente,-1,1);
68      [Norm_V,Param_V] = mapminmax(Voltaje,-1,1);
69
70      %---------------------Entrenamiento y simulación----------------------
71      p = [Norm_R;Norm_T];
72      target = [Norm_C; Norm_V];
73
74      net.trainParam.show = 25;
75      net.trainParam.epochs = 1500;
76      net.trainParam.goal = 0;
77      net.trainParam.max_fail = 6;
78      net.trainParam.min_grad = 1e-10;
79
80      [net,tr] = train(net,p,target);
81
82      Outputs = net(p);
83      trOut = Outputs(tr.trainInd);
84      vOut = Outputs(tr.valInd);
85      tsOut = Outputs(tr.testInd);
86
87      trTarg = target(tr.trainInd);
88      vTarg = target(tr.valInd);
89      tsTarg = target(tr.testInd);
90  plotregression(trTarg,trOut,'Train',vTarg,vOut,'Validation',tsTarg,tsOut,'Testing');
91
92      %-------------------Pesos y Bias-----------------------------
93
94      W_capa1 = net.iw{1,1};
95      Bias_capa1 = net.b{1};
96
97      W_capa2 = net.lw{2,1};
98      Bias_capa2 = net.b{2};
99
100     W_capa3 = net.lw{3,2};
101     Bias_capa3 = net.b{3};
```

Apéndice D.

Sección 1 Módulos de una Neurona Digital.

1. ROM_1 (ROM_1.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3     use IEEE.STD_LOGIC_ARITH.ALL;
4     use IEEE.STD_LOGIC_SIGNED.ALL;
5
6     entity ROM_1 is
7     generic (constant x1:integer:= 0;
8              constant x2:integer:= 0;
9              constant Size: Natural := 18);
10
11    port(adr1: in std_logic;
12         d: out std_logic_vector(size-1 downto 0));
13    end ROM_1;
14
15    architecture Behavioral of ROM_1 is
16    begin
17      process (adr1)
18       begin
19        case adr1 is
20          when '0' => d <= CONV_std_logic_vector(x1,Size);
21          when others => d <= CONV_std_logic_vector(x2,Size);
22        end case;
23    end process;
24    end Behavioral;
```

2. Unidad de almacenamiento MAC (MAC.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3     use IEEE.STD_LOGIC_ARITH.ALL;
4     use IEEE.STD_LOGIC_SIGNED.ALL;
5
6     entity MAC is
7     Port (X, Y, B : in std_logic_vector(17 downto 0);
8                 LOAD, CLK, CE : in std_logic;
9                 A : out std_logic_vector(35 downto 0));
10    end MAC;
11
12    architecture Behavioral of MAC is
13
14    signal ACC  :std_logic_vector(35 downto 0) := (others => '0');
15    signal XY   :std_logic_vector(35 downto 0) := (others => '0');
16    signal BIAS :std_logic_vector(35 downto 0) := (others => '0');
17
18    begin
19
```

```
20    BIAS <= O"777"&B&O"000" when B(17)='1' else O"000"&B&O"000";
21    XY <= X*Y;
22    A <= ACC;
23
24    process(CLK,LOAD,ACC,XY,BIAS,CE)
25          begin
26          if falling_edge(CLK) and (CE='1') then
27                if (LOAD = '1') then
28                      ACC <= BIAS;
29                else
30                      ACC <= ACC + XY;
31                end if;
32          end if;
33    end process;
34    end Behavioral;
```

3. Función TANSIG (Funcion_Activacion.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3     use IEEE.STD_LOGIC_ARITH.ALL;
4     use IEEE.STD_LOGIC_SIGNED.ALL;
5
6     entity TANSIG is
7     Port (X : in std_logic_vector(17 downto 0);
8                 Y : out std_logic_vector(17 downto 0);
9                 Enable : in std_logic);
10    end TANSIG;
11
12    architecture Arq of TANSIG is
13
14    constant a0:integer:=508; constant a1:integer:=473;
15    constant a2:integer:=414; constant a3:integer:=342;
16    constant a4:integer:=270; constant a5:integer:=205;
17    constant a6:integer:=152; constant a7:integer:=110;
18    constant a8:integer:=78;  constant a9:integer:=55;
19    constant aa:integer:=39;  constant ab:integer:=27;
20    constant ac:integer:=19;  constant ad:integer:=13;
21    constant ae:integer:=7;  constant af:integer:=1;
22
23    constant b0:integer:=0;   constant b1:integer:=7;
24    constant b2:integer:=29;  constant b3:integer:=70;
25    constant b4:integer:=124; constant b5:integer:=185;
26    constant b6:integer:=244; constant b7:integer:=299;
27    constant b8:integer:=347; constant b9:integer:=386;
28    constant ba:integer:=416; constant bb:integer:=441;
29    constant bc:integer:=459; constant bd:integer:=473;
30    constant be:integer:=492; constant bf:integer:=508;
31
32    constant x0:integer:=0;    constant x1:integer:=96;
33    constant x2:integer:=192;  constant x3:integer:=288;
34    constant x4:integer:=384;  constant x5:integer:=480;
35    constant x6:integer:=576;  constant x7:integer:=672;
36    constant x8:integer:=768;  constant x9:integer:=864;
37    constant xa:integer:=960;  constant xb:integer:=1056;
```

```
38    constant xc:integer:=1152; constant xd:integer:=1248;
39    constant xe:integer:=1344; constant xf:integer:=1440;
40    constant x10:integer:=1792;
41
42    signal abs_x:     std_logic_vector(17 downto 0);
43    signal latch:     std_logic_vector(17 downto 0);
44    signal Y_S:       std_logic_vector(17 downto 0);
45    signal abs_y:     std_logic_vector(35 downto 0);
46    signal a,b:       std_logic_vector(17 downto 0);
47
48
49    begin
50
51    abs_x<=abs(X);
52
53    process(abs_x)
54    begin
55
56          if (abs_x>= X0)and(abs_x< X1) then
57                a<=conv_std_logic_vector(a0,18);
58                b<=conv_std_logic_vector(b0,18);
59          elsif (abs_x< X2) then
60                a<=conv_std_logic_vector(a1,18);
61                b<=conv_std_logic_vector(b1,18);
62          elsif (abs_x< X3) then
63                a<=conv_std_logic_vector(a2,18);
64                b<=conv_std_logic_vector(b2,18);
65          elsif (abs_x< X4) then
66                a<=conv_std_logic_vector(a3,18);
67                b<=conv_std_logic_vector(b3,18);
68          elsif (abs_x< X5) then
69                a<=conv_std_logic_vector(a4,18);
70                b<=conv_std_logic_vector(b4,18);
71          elsif (abs_x< X6) then
72                a<=conv_std_logic_vector(a5,18);
73                b<=conv_std_logic_vector(b5,18);
74          elsif (abs_x< X7) then
75                a<=conv_std_logic_vector(a6,18);
76                b<=conv_std_logic_vector(b6,18);
77          elsif (abs_x< X8) then
78                a<=conv_std_logic_vector(a7,18);
79                b<=conv_std_logic_vector(b7,18);
80          elsif (abs_x< X9) then
81                a<=conv_std_logic_vector(a8,18);
82                b<=conv_std_logic_vector(b8,18);
83          elsif (abs_x< XA) then
84                a<=conv_std_logic_vector(a9,18);
85                b<=conv_std_logic_vector(b9,18);
86          elsif (abs_x< XB) then
87                a<=conv_std_logic_vector(aa,18);
88                b<=conv_std_logic_vector(ba,18);
89          elsif (abs_x< XC) then
90                a<=conv_std_logic_vector(ab,18);
91                b<=conv_std_logic_vector(bb,18);
92          elsif (abs_x< XD) then
93                a<=conv_std_logic_vector(ac,18);
```

```
94              b<=conv_std_logic_vector(bc,18);
95        elsif (abs_x< XE) then
96              a<=conv_std_logic_vector(ad,18);
97              b<=conv_std_logic_vector(bd,18);
98        elsif (abs_x< XF) then
99              a<=conv_std_logic_vector(ae,18);
100             b<=conv_std_logic_vector(be,18);
101       elsif (abs_x< X10)then
102             a<=conv_std_logic_vector(af,18);
103             b<=conv_std_logic_vector(bf,18);
104       else
105             a<=(others=>'0');
106             b<=conv_std_logic_vector(512,18);
107       end if;
108  end process;
109
110  abs_y<=(a*abs_x)+(b&O"000");
111
112  process(X,abs_y)
113  begin
114       if X(17)='1' then
115             Y_S <= O"000000"-abs_y(26 downto 9);
116       else
117             Y_S <= abs_y(26 downto 9);
118       end if;
119  end process;
120
121  latch <= Y_S when(Enable = '1') else latch;
122  Y <= latch;

124  end Arq;
```

Sección 2 Integración de la Neurona digital 1.

1. Neurona 1_1 (Nerona_1.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3     use IEEE.STD_LOGIC_ARITH.ALL;
4     use IEEE.STD_LOGIC_SIGNED.ALL;
5     use work.Pack_Red.all;
6
7     entity Neurona_1 is
8
9       generic (constant W1  :integer :=0;
10              constant W2  :integer :=0;
11              constant Bias:integer :=0);
12
13       Port (X1: in std_logic_vector(17 downto 0);
14             sal1: out std_logic_vector(17 downto 0);
15           LOAD, CLK, Enable, adr1,CE: in std_logic);
16
17    end Neurona_1;
18
19     architecture Behavioral of Neurona_1 is
```

```
20
21        signal W:std_logic_vector(17 downto 0);
22        signal A:std_logic_vector(17 downto 0);
23        signal E:std_logic_vector(17 downto 0);
24        signal B:std_logic_vector(17 downto 0);
25
26     begin
27            B <= CONV_std_logic_vector(Bias,18);
28    -- Llamado de los componentes

29    U1:ROM_1
30    generic map(x1=>W1, x2=>W2, size=>18)
31    port map (d=>W, adr1=>adr1);
32
33    U2:MAC port map (X=>X1, Y=>W, B=>B, LOAD=>LOAD, CLK=>CLK, CE=>CE, A(26
34    downto9)=>A,A(35downto27)=>E(8 downto 0),A(8 downto 0)=>E(17downto 9));
35
36    U3:TANSIG port map (X=>A, Enable=>Enable, Y=>sal1);
37      end Behavioral;
```

Sección 3 Módulos que conforman a la Red Neuronal Simple.

1. ROM_2 (ROM_2.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3     use IEEE.STD_LOGIC_ARITH.ALL;
4     use IEEE.STD_LOGIC_SIGNED.ALL;
5
6     entity ROM_2 is
7     generic (constant x1:integer:= 0;
8              constant x2:integer:= 0;
9              constant x3:integer:= 0;
10             constant x4:integer:= 0;
11             constant x5:integer:= 0;
12             constant x6:integer:= 0;
13             constant x7:integer:= 0;
14             constant Size: Natural := 18);
15
16    port(adr2: in std_logic_vector (2 downto 0);
17           d: out std_logic_vector (size-1 downto 0));
18    end ROM_2;
19
20    architecture Behavioral of ROM_2 is
21    begin
22      process (adr2)
23       begin
24        case adr2 is
25          when "000" => d <= CONV_std_logic_vector(x1,Size);
26          when "001" => d <= CONV_std_logic_vector(x2,Size);
27          when "010" => d <= CONV_std_logic_vector(x3,Size);
28          when "011" => d <= CONV_std_logic_vector(x4,Size);
29          when "100" => d <= CONV_std_logic_vector(x5,Size);
```

```
30          when "101" => d <= CONV_std_logic_vector(x6,Size);
31          when "110" => d <= CONV_std_logic_vector(x7,Size);
32          when others => d <= CONV_std_logic_vector('0',Size);
33     end case;
34     end process;
35  end Behavioral;
```

2. ROM_3 (ROM_3.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3     use IEEE.STD_LOGIC_ARITH.ALL;
4     use IEEE.STD_LOGIC_SIGNED.ALL;
5
6     entity ROM_3 is
7     generic (constant x1:integer:= 0;
8              constant x2:integer:= 0;
9              constant x3:integer:= 0;
10             constant x4:integer:= 0;
11             constant x5:integer:= 0;
12             constant x6:integer:= 0;
13             constant x7:integer:= 0;
14             constant x8:integer:= 0;
15             constant x9:integer:= 0;
16             constant Size: Natural := 18);
17
18    port(adr3: in std_logic_vector (3 downto 0);
19            d: out std_logic_vector (size-1 downto 0));
20    end ROM_3;
21
22    architecture Behavioral of ROM_3 is
23    begin
24      process (adr3)
25       begin
26        case adr3 is
27          when "0000" => d <= CONV_std_logic_vector(x1,Size);
28          when "0001" => d <= CONV_std_logic_vector(x2,Size);
29          when "0010" => d <= CONV_std_logic_vector(x3,Size);
30          when "0011" => d <= CONV_std_logic_vector(x4,Size);
31          when "0100" => d <= CONV_std_logic_vector(x5,Size);
32          when "0101" => d <= CONV_std_logic_vector(x6,Size);
33          when "0110" => d <= CONV_std_logic_vector(x7,Size);
34          when "0111" => d <= CONV_std_logic_vector(x8,Size);
35          when "1000" => d <= CONV_std_logic_vector(x9,Size);
36          when others => d <= CONV_std_logic_vector('0',Size);
37    end case;
38  end process;
39 end Behavioral;
```

3. Neurona 2 (Neurona_2.vhd)

```
1     use IEEE.STD_LOGIC_1164.ALL;
2     use IEEE.STD_LOGIC_ARITH.ALL;
3     use IEEE.STD_LOGIC_SIGNED.ALL;
```

```
4     use work.Pack_Red.all;
5
6     entity Neurona_2 is
7       generic (constant W1:integer :=0;
8                 constant W2:integer :=0;
9                 constant W3:integer :=0;
10                constant W4:integer :=0;
11                constant W5:integer :=0;
12                constant W6:integer :=0;
13                constant W7:integer :=0;
14                constant Bias: integer:=0);

15       Port (X2: in std_logic_vector(17 downto 0);
16             sal2: out std_logic_vector(17 downto 0);
17             adr2: in std_logic_vector (2 downto 0);
18             LOAD,CLK,Enable,CE: in std_logic);
19    end Neurona_2;
20
21    architecture Behavioral of Neurona_2 is
22    signal W:std_logic_vector(17 downto 0);
23    signal A:std_logic_vector(17 downto 0);
24    signal E:std_logic_vector(17 downto 0);
25    signal B:std_logic_vector(17 downto 0);
26
27 begin
28   B <= CONV_std_logic_vector(Bias,18);
29
30 -- Llamado de los componentes
31 U1:ROM_2
32 generic map(x1=>W1,x2=>W2,x3=>W3,x4=>W4,x5=>W5,x6=>W6,x7=>W7,size=>18)
33 port map (d=>W, adr2=>adr2);
34
35 U2:MAC port map(X=>X2,Y=>W,B=>B,LOAD=>LOAD,CLK=>CLK,CE=>CE,A(26downto9)=>A,
36 A(35 downto 27)=>E(8 downto 0), A(8 downto 0)=>E(17 downto 9));
37
38 U3:TANSIG port map (X=>A,Enable=>Enable,Y=>sal2);
39
40 end Behavioral;
```

4. Neurona 3 (Neurona_3.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3     use IEEE.STD_LOGIC_ARITH.ALL;
4     use IEEE.STD_LOGIC_SIGNED.ALL;
5     use work.Pack_Red.all;
6
7     entity Neurona_3 is
8      generic (constant W1:integer :=0;
9                constant W2:integer :=0;
10               constant W3:integer :=0;
11               constant W4:integer :=0;
12               constant W5:integer :=0;
13               constant W6:integer :=0;
14               constant W7:integer :=0;
```

```
15              constant W8:integer :=0;
16              constant W9:integer :=0;
17              constant Bias: integer:=0);
18
19    Port (X3: in std_logic_vector(17 downto 0);
20          sal3: out std_logic_vector(17 downto 0);
21          adr3: in std_logic_vector (3 downto 0);
22          LOAD, CLK, Enable,CE : in std_logic);
23    end Neurona_3;
24
25    architecture Behavioral of Neurona_3 is
26     signal W:std_logic_vector(17 downto 0);
27    signal A:std_logic_vector(17 downto 0);
28    signal E:std_logic_vector(17 downto 0);
29    signal B:std_logic_vector(17 downto 0);
30
31 begin
32  B <= CONV_std_logic_vector(Bias,18);
33
34 -- Llamado de los componentes
35 U1:ROM_3 generic map(x1=>W1,x2=>W2,x3=>W3,x4=>W4,x5=>W5,x6=>W6,x7=>W7,
36 x8=>W8, x9=>W9, size=>18)
37 port map (d=>W, adr3=>adr3);
38
39 U2:MAC port map (X=>X3,Y=>W,B=>B,LOAD=>LOAD,CLK=>CLK,CE=>CE,A(26downto9)=>A,
40 A(35 downto 27)=>E(8 downto 0), A(8 downto 0)=>E(17 downto 9));
41
42 U3:TANSIG port map (X=>A, Enable=>Enable, Y=>sal3);
43
44 end Behavioral;
```

5. Multiplexor 2:1 (Mux_2X1.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3
4     entity MUX1_2x1 is
5
6     port (H: in std_logic_vector (17 downto 0);
7           T: in std_logic_vector (17 downto 0);
8         Sal: out std_logic_vector (17 downto 0);
9            SEL1: in std_logic);
10   end MUX1_2x1;
11
12   architecture Behavioral of MUX1_2x1 is
13     begin
14         with SEL1 select
15              Sal <= H when '0',
16                     T when others;
18     end Behavioral;
```

6. Multiplexor 8:1 (Mux_8X1.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3
4     entity MUX2_8x1 is
5
6     port (E0, E1, E2, E3, E4, E5, E6, E7: in std_logic_vector (17 downto 0);
7           SEL2: in std_logic_vector (2 downto 0);
8            Sal: out std_logic_vector (17 downto 0));
9     end MUX2_8x1;
10
11    architecture Behavioral of MUX2_8x1 is
12      begin
13         with SEL2 select
14              Sal <= E0 when "000",
15                     E1 when "001",
16                     E2 when "010",
17                     E3 when "011",
18                     E4 when "100",
19                     E5 when "101",
20                     E6 when "110",
21                     E7 when others;
22    end Behavioral;
```

7. Multiplexor 10:1 (Mux_10X1.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3
4     entity MUX3_10x1 is
5
6     port (E0,E1,E2,E3,E4,E5,E6,E7,E8,E9: in std_logic_vector (17 downto 0);
7                  SEL3: in std_logic_vector (3 downto 0);
8             Sal: out std_logic_vector (17 downto 0));
9     end MUX3_10x1;
10
11    architecture Behavioral of MUX3_16x1 is
12      begin
13         with SEL3 select
14              Sal <= E0 when "0000",
15                     E1 when "0001",
16                     E2 when "0010",
17                     E3 when "0011",
18                     E4 when "0100",
19                     E5 when "0101",
20                     E6 when "0110",
21                     E7 when "0111",
22                     E8 when "1000",
23                     E9 when others;
24      end Behavioral;
```

8. Red Neuronal (Red_Neuronal.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3     use IEEE.STD_LOGIC_ARITH.ALL;
4     use IEEE.STD_LOGIC_SIGNED.ALL;
5     use work.Pack_Red.all;
6
7     entity Red_Neuronal_Pipeline is
8     ----------------------- DECLARACIÓN DE PESOS Y BIAS
9  generic (constant W11_1:integer:=1671; constant W12_1:integer:=262070;
10 constant B1_1: integer:=260126;
11 constant W21_1:integer:=684; constant W22_1:integer:=1381; constant B2_1:
12 integer:=260700;
13 constant W31_1:integer:=262046; constant W32_1:integer:=261593; constant
14 B3_1: integer:=261861;
15 constant W41_1:integer:=788; constant W42_1:integer:=261839; constant B4_1:
16 integer:=261816;
17 constant W51_1:integer:=260738; constant W52_1:integer:=415; constant B5_1:
18 integer:=261151;
19 constant W61_1:integer:=881; constant W62_1:integer:=925; constant B6_1:
20 integer:=1158;
21 constant W71_1:integer:=502; constant W72_1:integer:=1491; constant B7_1:
22 integer:=1957;
23
24 constant W11_2:integer:=261971; constant W12_2:integer:=262020; constant
25 W13_2:integer:=1132; constant W14_2:integer:=53;
26 constant W15_2:integer:=283; constant W16_2:integer:=374; constant
27 W17_2:integer:=261983; constant B1_2:integer:=1088;
28 constant W21_2:integer:=613; constant W22_2:integer:=124; constant
29 W23_2:integer:=262066; constant W24_2:integer:=348;
30 constant W25_2:integer:=261465; constant W26_2:integer:=404; constant
31 W27_2:integer:=261789; constant B2_2:integer:=261468;
32 constant W31_2:integer:=218; constant W32_2:integer:=242; constant
33 W33_2:integer:=564; constant W34_2:integer:=262131;
34 constant W35_2:integer:=262140; constant W36_2:integer:=46; constant
35 W37_2:integer:=261720; constant B3_2:integer:=448;
36 constant W41_2:integer:=261497; constant W42_2:integer:=120; constant
37 W43_2:integer:=165; constant W44_2:integer:=382;
38 constant W45_2:integer:=261739; constant W46_2:integer:=131; constant
39 W47_2:integer:=476; constant B4_2:integer:=296;
40 constant W51_2:integer:=262003; constant W52_2:integer:=262096; constant
41 W53_2:integer:=262042; constant W54_2:integer:=142;
42 constant W55_2:integer:=261207; constant W56_2:integer:=164; constant
43 W57_2:integer:=805; constant B5_2:integer:=13;
44 constant W61_2:integer:=262114; constant W62_2:integer:=968; constant
45 W63_2:integer:=261244; constant W64_2:integer:=261832;
46 constant W65_2:integer:=250; constant W66_2:integer:=261884; constant
47 W67_2:integer:=261971; constant B6_2:integer:=261523
48 constant W71_2:integer:=663; constant W72_2:integer:=241; constant
49 W73_2:integer:=262113; constant W74_2:integer:=261462;
50 constant W75_2:integer:=239; constant W76_2:integer:=262050; constant
51 W77_2:integer:=262140; constant B7_2:integer:=330
52 constant W81_2:integer:=364; constant W82_2:integer:=54; constant
53 W83_2:integer:=261948; constant W84_2:integer:=213;
54 constant W85_2:integer:=35; constant W86_2:integer:=261696; constant
```

```
55 W87_2:integer:=261914; constant B8_2:integer:=260810;
56 constant W91_2:integer:=489; constant W92_2:integer:=261986; constant
57 W93_2:integer:=604; constant W94_2:integer:=262108;
58 constant W95_2:integer:=261810; constant W96_2:integer:=261687; constant
59 W97_2:integer:=401; constant B9_2:integer:=904;
60
61 constant W11_3:integer:=262086; constant W12_3:integer:=1374; constant
62 W13_3:integer:=262127; constant W14_3:integer:=262126;
63 constant W15_3:integer:=859; constant W16_3:integer:=262047; constant
64 W17_3:integer:=261678; constant W18_3:integer:=481;
65 constant W19_3:integer:=261771; constant B1_3:integer:=818;
66 constant W21_3:integer:=444; constant W22_3:integer:=261980; constant
67 W23_3:integer:=527; constant W24_3:integer:=136;
68 constant W25_3:integer:=6; constant W26_3:integer:=261611; constant
69 W27_3:integer:=78; constant W28_3:integer:=776;
70 constant W29_3:integer:=390; constant B2_3:integer:=261895);
71
72  port (H, T: in std_logic_vector (17 downto 0);
73        I, V: out std_logic_vector (17 downto 0);
74        SEL1,LOAD,clk, Enable,CE1,CE2,CE3: in std_logic;
75        SEL2: in std_logic_vector (2 downto 0);
76        SEL3: in std_logic_vector (3 downto 0));
77
78    end Red_Neuronal;
79
80  architecture Behavioral of Red_Neuronal is
81
82 signal cm_x1, cm_x2, cm_x3: std_logic_vector (17 downto 0);
83 signal c1_0,c1_1,c1_2,c1_3,c1_4,c1_5,c1_6,c1_7:std_logic_vector(17downto0);
84 signal c2_0,c2_1,c2_2,c2_3,c2_4,c2_5,c2_6,c2_7,c2_8,c2_9: std_logic_vector
85 (17 downto 0);
86
87    begin
88
89    ------------------MULTIPLEXORES-----------------------
90
91    U1:MUX1_2x1 port map (H=>H, T=>T,SEL1=>SEL1, Sal=>cm_x1);
92    U2:MUX2_8x1 port map (E0=>c1_0, E1=>c1_1, E2=>c1_2, E3=>c1_3, E4=>c1_4,
93    E5=>c1_5, E6=>c1_6, E7=>c1_7,SEL2=>SEL2, Sal=>cm_x2);
94    U3:MUX3_10x1 port map(E0=>c2_0, E1=>c2_1, E2=>c2_2, E3=>c2_3, E4=>c2_4,
95    E5=>c2_5,E6=>c2_6, E7=>c2_7,E8=>c2_8, E9=>c2_9, SEL3=>SEL3, Sal=>cm_x3);
96
97    ------------------   CAPA1  -----------------------------
98
99    U4:Neurona_1
100   generic map(W1=>W11_1, W2=>W12_1, Bias=>B1_1)
101   port map (X1=>cm_x1, adr1=>SEL1, LOAD=>LOAD, clk=>clk, Enable=>Enable,
102   CE=>CE1, sal1=>c1_0);
103
104   U5:Neurona_1
105   generic map(W1=>W21_1, W2=>W22_1, Bias=>B2_1)
106   port map (X1=>cm_x1, adr1=>SEL1, LOAD=>LOAD, clk=>clk, Enable=>Enable,
107   CE=>CE1, sal1=>c1_1);
108
109   U6:Neurona_1
110   generic map(W1=>W31_1, W2=>W32_1, Bias=>B3_1)
```

```
111   port map (X1=>cm_x1, adr1=>SEL1, LOAD=>LOAD, clk=>clk, Enable=>Enable,
112   CE=>CE1, sal1=>c1_2);
113
114   U7:Neurona_1
115   generic map(W1=>W41_1, W2=>W42_1, Bias=>B4_1)
116   port map (X1=>cm_x1, adr1=>SEL1, LOAD=>LOAD, clk=>clk, Enable=>Enable,
117   CE=>CE1, sal1=>c1_3);
118
119   U8:Neurona_1
120   generic map(W1=>W51_1, W2=>W52_1, Bias=>B5_1)
121   port map (X1=>cm_x1, adr1=>SEL1, LOAD=>LOAD, clk=>clk, Enable=>Enable,
122   CE=>CE1, sal1=>c1_4);
123
124   U9:Neurona_1
125   generic map(W1=>W61_1, W2=>W62_1, Bias=>B6_1)
126   port map (X1=>cm_x1, adr1=>SEL1, LOAD=>LOAD, clk=>clk, Enable=>Enable,
127   CE=>CE1, sal1=>c1_5);
128
129   U10:Neurona_1
130   generic map(W1=>W71_1, W2=>W72_1, Bias=>B7_1)
131   port map (X1=>cm_x1, adr1=>SEL1, LOAD=>LOAD, clk=>clk, Enable=>Enable,
132   CE=>CE1, sal1=>c1_6);
133
134   ------------------   CAPA2  ------------------------------
135   U11:Neurona_2
136   generic map(W1=>W11_2, W2=>W12_2, W3=>W13_2, W4=>W14_2, W5=>W15_2,
137   W6=>W16_2, W7=>W17_2, Bias=>B1_2)
138   port map (X2=>cm_x2, adr2=>SEL2, LOAD=>LOAD, clk=>clk, Enable=>Enable,
139   CE=>CE2, sal2=>c2_0);
140
141   U12:Neurona_2
142   generic map(W1=>W21_2, W2=>W22_2, W3=>W23_2, W4=>W24_2, W5=>W25_2,
143   W6=>W26_2, W7=>W27_2, Bias=>B2_2)
144   port map (X2=>cm_x2, adr2=>SEL2, LOAD=>LOAD, clk=>clk, Enable=>Enable,
145   CE=>CE2, sal2=>c2_1);
146
147   U13:Neurona_2
148   generic map(W1=>W31_2, W2=>W32_2, W3=>W33_2, W4=>W34_2, W5=>W35_2,
149   W6=>W36_2, W7=>W37_2, Bias=>B3_2)
150   port map (X2=>cm_x2, adr2=>SEL2, LOAD=>LOAD, clk=>clk, Enable=>Enable,
151   CE=>CE2, sal2=>c2_2);
152
153   U14:Neurona_2
154   generic map(W1=>W41_2, W2=>W42_2, W3=>W43_2, W4=>W44_2, W5=>W45_2,
155   W6=>W46_2, W7=>W47_2, Bias=>B4_2)
156   port map (X2=>cm_x2, adr2=>SEL2, LOAD=>LOAD, clk=>clk, Enable=>Enable,
157   CE=>CE2, sal2=>c2_3);
158
159   U15:Neurona_2
160   generic map(W1=>W51_2, W2=>W52_2, W3=>W53_2, W4=>W54_2, W5=>W55_2,
161   W6=>W56_2, W7=>W57_2, Bias=>B5_2)
162   port map (X2=>cm_x2, adr2=>SEL2, LOAD=>LOAD, clk=>clk, Enable=>Enable,
163   CE=>CE2, sal2=>c2_4);
164
165   U16:Neurona_2
166   generic map(W1=>W61_2, W2=>W62_2, W3=>W63_2, W4=>W64_2, W5=>W65_2,
167   W6=>W66_2, W7=>W67_2, Bias=>B6_2)
```

```
168   port map (X2=>cm_x2, adr2=>SEL2, LOAD=>LOAD, clk=>clk, Enable=>Enable,
169   CE=>CE2, sal2=>c2_5);
170
171   U17:Neurona_2
172   generic map(W1=>W71_2, W2=>W72_2, W3=>W73_2, W4=>W74_2, W5=>W75_2,
173   W6=>W76_2, W7=>W77_2, Bias=>B7_2)
174   port map (X2=>cm_x2, adr2=>SEL2, LOAD=>LOAD, clk=>clk, Enable=>Enable,
175   CE=>CE2, sal2=>c2_6);
176
177   U18:Neurona_2
178   generic map(W1=>W81_2, W2=>W82_2, W3=>W83_2, W4=>W84_2, W5=>W85_2,
179   W6=>W86_2, W7=>W87_2, Bias=>B8_2)
180   port map (X2=>cm_x2, adr2=>SEL2, LOAD=>LOAD, clk=>clk, Enable=>Enable,
181   CE=>CE2, sal2=>c2_7);
182
183   U19:Neurona_2
184   generic map(W1=>W91_2, W2=>W92_2, W3=>W93_2, W4=>W94_2, W5=>W95_2,
185   W6=>W96_2, W7=>W97_2, Bias=>B9_2)
186   port map (X2=>cm_x2, adr2=>SEL2, LOAD=>LOAD, clk=>clk, Enable=>Enable,
187   CE=>CE2, sal2=>c2_8);
188
189------------------   CAPA3  ------------------------------
190
191   U20:Neurona_3
192   generic map(W1=>W11_3, W2=>W12_3, W3=>W13_3, W4=>W14_3, W5=>W15_3,
193   W6=>W16_3, W7=>W17_3, W8=>W18_3, W9=>W19_3, Bias=>B1_3)
194   port map (X3=>cm_x3, adr3=>SEL3, LOAD=>LOAD, clk=>clk, Enable=>Enable,
195   CE=>CE3, sal3=>I);
196
197   U21:Neurona_3
198   generic map(W1=>W21_3, W2=>W22_3, W3=>W23_3, W4=>W24_3, W5=>W25_3,
199   W6=>W26_3, W7=>W27_3, W8=>W28_3, W9=>W29_3, Bias=>B2_3)
200   port map (X3=>cm_x3, adr3=>SEL3, LOAD=>LOAD, clk=>clk, Enable=>Enable,
201   CE=>CE3, sal3=>V);
202
203   end Behavioral;
```

Sección 5. Módulos que conforman a la Red Neuronal con técnica Pipeline.

1. Controlador (Controlador.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3     use IEEE.STD_LOGIC_ARITH.ALL;
4     use IEEE.STD_LOGIC_SIGNED.ALL;
5     use work.Pack_Red.all;
6
7     entity controlador is
8         port (rst, clk: in std_logic;
9               clk_out: out std_logic;
10                 sal: out std_logic_vector (8 downto 0));
11    end controlador;
12
13    architecture Behavioral of controlador is
14
```

```
15    signal clk_10, clk_20, locked: std_logic;
16
17    begin
18
19    U_DCM : clk_div port map( clk, clk_10, clk_20, rst, locked);
20
21    clk_out <= clk_10;
22
23    process (clk_20, locked)

25       variable aux : std_logic_vector (5 downto 0);
24
27        begin
28
29        if locked = '0' or aux = "011000" then
30         aux := (others => '0');
31
32          elsif (clk_20'event and clk_20 = '1') then
33          aux := aux + 1;
34          case aux is
35
36          when "000001" => sal <= '1' & X"70";
37          when "000010" => sal <= '1' & X"70";
38          when "000011" => sal <= '0' & X"70";
39          when "000100" => sal <= '0' & X"70";
40          when "000101" => sal <= '0' & X"70";
41          when "000110" => sal <= '0' & X"71";
42          when "000111" => sal <= '0' & X"31";
43          when "001000" => sal <= '0' & X"32";
44          when "001001" => sal <= '0' & X"32";
45          when "001010" => sal <= '0' & X"33";
46          when "001011" => sal <= '0' & X"33";
47          when "001100" => sal <= '0' & X"34";
48          when "001101" => sal <= '0' & X"34";
49          when "001110" => sal <= '0' & X"35";
50          when "001111" => sal <= '0' & X"35";
51          when "010000" => sal <= '0' & X"36";
52          when "010001" => sal <= '0' & X"16";
53          when "010010" => sal <= '0' & X"17";
54          when "010011" => sal <= '0' & X"17";
55          when "010100" => sal <= '0' & X"18";
56          when "010101" => sal <= '0' & X"08";
57          when "010110" => sal <= '0' & X"00";
58          when "010111" => sal <= '0' & X"80";
59          when others   => sal <= '0' & X"00";
60
61          end case;
62       end if;
63    end process;
64  end Behavioral;
```

2. Bloque de Entrada (Entrada.vhd)

```
1     ibrary IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3     use IEEE.STD_LOGIC_ARITH.ALL;
4     use IEEE.STD_LOGIC_SIGNED.ALL;

6     entity Entrada is
7           Port(clk,rst: in STD_LOGIC;
8                     H,T: out STD_LOGIC_VECTOR(17 downto 0));
9     end Entrada;
10
11    architecture Arq of Entrada is
12    signal address : integer:=0;
13
14    type ram_t is array (0 to 164) of std_logic_vector(0 to 19);
15
16    signal RAD: ram_t:= (
17    --DATOS DE RADIACION DE PRUEBA
18    X"FFE5E",X"00049",X"FFF28",X"FFF62",X"00023",X"FFEE9",X"000DE",X"00115",X"00163",X"0017E",
19    X"0018C",X"00196",X"000FD",X"00104",X"00160",X"0010A",X"0018F",X"00160",X"0015C",X"00141",
20    X"001B8",X"001A4",X"0014F",X"00133",X"00118",X"00141",X"00182",X"FFEDE",X"FFFF4",X"FFFEE",
21    X"0005A",X"0002A",X"FFEAC",X"0017B",X"00126",X"00082",X"000EF",X"0004C",X"0011F",X"00089",
22    X"00071",X"00122",X"000A1",X"00071",X"00008",X"FFFB7",X"FFFBE",X"FFFB7",X"FFF92",X"FFF7D",
23    X"FFF7A",X"FFF6C",X"FFF7A",X"FFF9F",X"FFFA6",X"FFFA3",X"FFFB0",X"FFFC5",X"FFFD9",X"FFFF1",
24    X"0002A",X"00023",X"00049",X"00045",X"0002A",X"00042",X"0003E",X"00049",X"00001",X"FFFE7",
25    X"00086",X"0004C",X"00064",X"0004F",X"0000F",X"0007F",X"00097",X"00086",X"00093",X"00093",
26    X"00060",X"00060",X"00034",X"00045",X"00026",X"00020",X"FFF03",X"FFFC5",X"00097",X"00086",
27    X"00163",X"0018C",X"000E5",X"00082",X"0014F",X"00129",X"FFEE3",X"00129",X"000D4",X"00064",
28    X"0000F",X"FFF77",X"FFF70",X"FFF51",X"FFF69",X"FFF73",X"FFEDE",X"FFF51",X"FFF3D",X"FFF62",
29    X"FFF03",X"FFFF1",X"000E5",X"00122",X"0013A",X"00159",X"0004C",X"0018F",X"0017E",X"001BF",
30    X"001DD",X"0019D",X"0018C",X"001BB",X"000B2",X"000CD",X"001AE",X"001C6",X"00115",X"00152",
31    X"00152",X"00118",X"0003B",X"0012D",X"0018F",X"FFFA6",X"00075",X"0014F",X"00188",X"0007C",
32    X"00049",X"FFFB0",X"000D4",X"00133",X"00064",X"00166",X"000D7",X"0006E",X"FFF88",X"FFFDD",
33    X"FFFEA",X"FFF77",X"000C0",X"FFFE0",X"0008D",X"00071",X"00060",X"0001C",X"FFFB4",X"FFFF4",
34    X"FFFAA",X"FFFA3",X"FFF92",X"FFF69",X"FFEF9");
35
36    signal TEM: ram_t:= (
37    --DATOS DE TEMPERATURA DE PRUEBA
38    X"FFE00",X"FFEB9",X"FFE90",X"FFE9F",X"FFEC4",X"FFE64",X"FFE89",X"FFE8C",X"FFF1C",X"FFEEC",
39    X"FFF0D",X"FFF0D",X"FFF5E",X"FFF4C",X"FFF23",X"FFF23",X"FFEF4",X"FFEDA",X"FFFAF",X"FFFAF",
40    X"FFF87",X"FFF5B",X"FFF66",X"FFFBE",X"FFFC9",X"FFFA8",X"FFFE3",X"FFF4C",X"FFF6D",X"FFF5E",
41    X"FFF92",X"FFFA8",X"FFF5E",X"FFFEA",X"FFFD8",X"FFFB7",X"FFFBE",X"FFF99",X"FFFD4",X"FFFC9",
42    X"FFF99",X"FFFBE",X"FFF2F",X"FFF48",X"FFE98",X"FFEE9",X"FFEF7",X"FFEC0",X"FFEDA",X"FFE43",
43    X"FFE55",X"FFE98",X"FFE7A",X"FFE73",X"FFE98",X"FFE55",X"FFE9F",X"FFE8C",X"FFEB9",X"FFEDA",
44    X"FFF02",X"FFF06",X"FFEBC",X"FFECF",X"FFF2B",X"FFF1C",X"FFF66",X"FFF50",X"FFEFF",X"FFEF0",
45    X"FFF23",X"FFF69",X"FFF5E",X"FFF7C",X"FFF5E",X"FFF8B",X"FFF5E",X"FFF8B",X"FFF7C",X"FFF7C",
46    X"FFF80",X"FFFDF",X"FFFD4",X"FFFE3",X"FFFAF",X"FFFDF",X"FFF71",X"FFF83",X"FFE89",X"FFE7E",
47    X"FFF87",X"FFF53",X"FFFE3",X"FFE94",X"FFF23",X"FFFC9",X"FFF83",X"FFFA8",X"FFFCD",X"FFF7C",
48    X"FFF27",X"FFEC0",X"FFEA3",X"FFF02",X"FFF87",X"FFE64",X"FFEFB",X"FFF23",X"FFF0A",X"FFE9B",
49    X"FFECF",X"FFECB",X"FFF1C",X"FFF48",X"FFF18",X"FFF18",X"FFF06",X"FFEDA",X"FFF15",X"FFF15",
50    X"FFF41",X"FFFA1",X"FFF87",X"FFF7C",X"FFF99",X"FFF8B",X"FFFA8",X"FFF96",X"FFFA1",X"FFFA1",
51    X"FFFBB",X"FFFA8",X"FFF5E",X"FFF8B",X"FFFB3",X"FFF69",X"FFF92",X"FFF99",X"FFFDC",X"FFF8B",
52    X"FFF8B",X"FFF66",X"FFFBE",X"FFFB7",X"FFF92",X"FFFA4",X"FFF96",X"FFFB7",X"FFF69",X"FFFBE",
53    X"FFFB7",X"FFF83",X"FFFC9",X"FFF69",X"FFFC2",X"FFF83",X"FFF4C",X"FFF71",X"FFF41",X"FFF66",
54    X"FFF41",X"FFF27",X"FFED2",X"FFED2",X"FFF06");
55
56    begin
57    process(clk)
58     begin
59        if(rising_edge(clk)) then
60            H <= RAD(address)(2 to 19);
61            T <= TEM(address)(2 to 19);
```

```
62        end if;
63     end process;
64
65    process(clk, rst) is
66     begin
67          if rst = '1' then
68          address<= 0;
69          elsif rising_edge (clk) then
70          if address < RAD'high then
71          address <= address + 1;
72          else
73          address <= address;
74         end if;
75       end if;
76    end process ;
77    end Arq;
```

3. Bloque de Salida (Salida.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3     use work.Pack_Red.all;
4
5     entity SALIDA is
6       port(tx: out std_logic;
7            I, V: in std_logic_vector (17 downto 0);
8            em, enable, we, clk_10, push, rst: in std_logic);
9     end SALIDA;
10
11    architecture Behavioral of SALIDA is
12    signal stop, señal2, señal, etx, eload: std_logic;
13    signal sal1, sal2: std_logic_vector (17 downto 0);
14
15    begin
16
17    ----------------------MEMORIA_WR-------------------------
18    US0: Memoria_WR port map (I=>I,V=>V,señal2=>señal2,señal=>señal,
19    rst=>rst,push=>push,we=>we,stop=>stop,Isal=>sal1,Vsal=>sal2);
20
21    ----------------------MUX1-----------------------------
22    US1: MUX port map (em=>em,eload=>eload,we=>we,señal=>señal);
23
24    ----------------------MUX2-----------------------------
25    US2: MUX2 port map (Enable=>enable, etx=>etx, we=>we, señal2=>señal2);
26
27    ----------------------SERIAL---------------------------
28    US3: SERIAL port map (I=>sal1,V=>sal2,eload=>eload,etx=>etx,stop=>stop,
29    clk=>clk_10, we=>we, tx=>tx);
30
31    end Behavioral;
```

4. Memoria (Memoria_WR.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3     use IEEE.STD_LOGIC_ARITH.ALL;
4     use IEEE.STD_LOGIC_SIGNED.ALL;
5     use work.Pack_Red.all;
6
7     entity Memoria_WR is
8       port(I, V: in std_logic_vector (17 downto 0);
9            señal2, señal, rst, push, we: in std_logic;
10                                  stop: inout std_logic;
11           Isal, Vsal: out std_logic_vector (17 downto 0));
12      end Memoria_WR;
13
14    architecture Behavioral of Memoria_WR is
15    signal address : integer:=0;
16
17    type ram_t is array(0 to 164) of STD_LOGIC_VECTOR(0 to 17);
18    signal Ipv: ram_t;
19    signal Vpv: ram_t;
20
21    begin
22    process (señal, we)
23      begin
24        if(rising_edge(señal)) then
25          if(we='1') then
26          Ipv(address) <= I;
27          Vpv(address) <= V;
28             else
29          Isal <= Ipv(address);
30          Vsal <= Vpv(address);
31          end if;
32        end if;
33      end process;
34
35    process(señal2, rst, push) is
36      begin
37         if (rst = '1' or push = '1') then
38         address <= 0;
39            stop <= '0';
40         elsif rising_edge (señal2) then
41         if address < Ipv'high then
42            address <= address + 1;
43         else
44            address <= address;
45               stop <= '1';
46         end if;
47         end if;
48    end process;
49    end Behavioral;
```

5. Multiplexor (Mux.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3     use work.Pack_Red.all;
4
5     entity MUX is
6       port (em, eload, we: in std_logic;
7             señal: out std_logic);
8     end MUX;
9
10    architecture Behavioral of MUX is
11    begin
12       with we select
13           señal <= em when '1',
14           eload when others;
15    end Behavioral;
```

6. Multiplexor 2 (Mux2.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3     use work.Pack_Red.all;
4
5     entity MUX2 is
6       port (Enable, etx, we: in std_logic;
7             señal2: out std_logic);
8     end MUX2;
9
10    architecture Behavioral of MUX2 is
11    begin
12      with we select
13        señal2 <= Enable when '1',
14                  etx when others;
15    end Behavioral;
```

7. Serial (Serial.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3     use IEEE.STD_LOGIC_ARITH.ALL;
4     use IEEE.STD_LOGIC_UNSIGNED.ALL;
5
6     entity SERIAL is
7     port (I, V: in STD_LOGIC_VECTOR(17 downto 0);
8           eload, etx: inout STD_LOGIC;
9           clk, we, stop: in STD_LOGIC;
10          tx: out STD_LOGIC);
11    end SERIAL;
12
13    architecture Behavioral of SERIAL is
14    signal regtx: STD_LOGIC_VECTOR  (7 downto 0) := (others =>'0');
```

```
15    signal cnttx: STD_LOGIC_VECTOR (15 downto 0) := (others =>'0');
16    signal   ttx: STD_LOGIC_VECTOR  (3 downto 0) := (others =>'0');
17    signal contador: STD_LOGIC_VECTOR (19 downto 0) := (others =>'0');
18    signal cargador: STD_LOGIC_VECTOR  (3 downto 0) := (others =>'0');
19    signal V1,V2,V3: STD_LOGIC_VECTOR  (7 downto 0) := (others =>'0');
20    signal I1,I2,I3: STD_LOGIC_VECTOR  (7 downto 0) := (others =>'0');
21
22    -------------- VELOCIDAD DE TRANSMISIÓN ----------------------------
23    ---Para 9600 baudios a 10MHz---
24    constant baudtx: STD_LOGIC_VECTOR(15 downto 0):="0000010000010010";
25
26    begin
27    -- Proceso para generar la señal eload y etx
28    process(we,contador,clk,stop)
29    begin
30    if we='1' then
31       eload<='1';
32       etx<='0';
33    elsif (clk'event and clk='1') then
34    if (contador <= X"00005" or stop='1')then
35        eload<='1';
36        etx<='0';
37    else
38        eload<='0';
39        etx<='1';
40    end if;
41    end if;
42   end process;
43
44    process(we,clk,contador)
45    begin
46    if (we='1' or contador=X"108DD")then
47        contador<=X"00000";
48    elsif (clk'event and clk='1') then
49    contador<=contador + 1;
50    end if;
51    end process;
52
53    -- Proceso para cargar los datos
54    process(we,eload,stop)
55    begin
56
57    if (we='1' or stop='1') then
58    V1<=(others=>'0');
59    V2<=(others=>'0');
60    V3<=(others=>'0');
61    I1<=(others=>'0');
62    I2<=(others=>'0');
63    I3<=(others=>'0');
64
65    elsif(eload'event and eload='0')then
66    V1<="000000"&V(17 downto 16);
67    V2<=V(15 downto 8);
68    V3<=V(7 downto 0);
69    I1<="000000"&I(17 downto 16);
69    I2<=I(15 downto 8);
70    I3<=I(7 downto 0);
```

```
71    end if;
72    end process;
73
74    -- Reloj de transmisión
75     process (clk,we,cnttx,etx)
76      begin
77          if (we='1' or etx='0')then
78           cnttx <= baudtx - 1;
79           ttx   <= "0000";
80           cargador <= "0000";
81          elsif (ttx = "1011") then
82           ttx <= "0000";
83          elsif (clk'event and clk='1' and etx='1')then
84           cnttx <= cnttx+1;
85          if (cnttx=baudtx)then
86           ttx<=ttx+1;
87           cnttx<=(others=>'0');
88          if (ttx="1010")then
89           cargador <= cargador + 1;
90          end if;
91          end if;
92         end if;
93    end process;
94
95    -- Asignacion de caracter
96    with cargador select
97          regtx <=    I1    when "0000",
98                      I2    when "0001",
99                      I3    when "0010",
100                     V1    when "0011",
101                     V2    when "0100",
102                     V3    when "0101",
103                     X"00" when others;

104   -- Protocolo de transmisión
105   with ttx select
106         tx <= '1' when "0000",
107               '0' when "0001",
108          regtx(0) when "0010",
109          regtx(1) when "0011",
110          regtx(2) when "0100",
111          regtx(3) when "0101",
112          regtx(4) when "0110",
113          regtx(5) when "0111",
114          regtx(6) when "1000",
115          regtx(7) when "1001",
116               '1' when "1010",
117               '1' when others;
118   end Behavioral;
```

8. Red Neuronal Pipeline (Red_Neuronal_Pipeline.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.ALL;
3     use IEEE.STD_LOGIC_ARITH.ALL;
4     use IEEE.STD_LOGIC_SIGNED.ALL;
5     use work.Pack_Red.all;
6
7     entity Red_Neuronal_Pipeline is
8     ----------------------- DECLARACIÓN DE PESOS Y BIAS
9  generic (constant W11_1:integer:=1671; constant W12_1:integer:=262070;
10 constant B1_1: integer:=260126;
11 constant W21_1:integer:=684; constant W22_1:integer:=1381; constant B2_1:
12 integer:=260700;
13 constant W31_1:integer:=262046; constant W32_1:integer:=261593; constant
14 B3_1: integer:=261861;
15 constant W41_1:integer:=788; constant W42_1:integer:=261839; constant B4_1:
16 integer:=261816;
17 constant W51_1:integer:=260738; constant W52_1:integer:=415; constant B5_1:
18 integer:=261151;
19 constant W61_1:integer:=881; constant W62_1:integer:=925; constant B6_1:
20 integer:=1158;
21 constant W71_1:integer:=502; constant W72_1:integer:=1491; constant B7_1:
22 integer:=1957;
23
24 constant W11_2:integer:=261971; constant W12_2:integer:=262020; constant
25 W13_2:integer:=1132; constant W14_2:integer:=53;
26 constant W15_2:integer:=283; constant W16_2:integer:=374; constant
27 W17_2:integer:=261983; constant B1_2:integer:=1088;
28 constant W21_2:integer:=613; constant W22_2:integer:=124; constant
29 W23_2:integer:=262066; constant W24_2:integer:=348;
30 constant W25_2:integer:=261465; constant W26_2:integer:=404; constant
31 W27_2:integer:=261789; constant B2_2:integer:=261468;
32 constant W31_2:integer:=218; constant W32_2:integer:=242; constant
33 W33_2:integer:=564; constant W34_2:integer:=262131;
34 constant W35_2:integer:=262140; constant W36_2:integer:=46; constant
35 W37_2:integer:=261720; constant B3_2:integer:=448;
36 constant W41_2:integer:=261497; constant W42_2:integer:=120; constant
37 W43_2:integer:=165; constant W44_2:integer:=382;
38 constant W45_2:integer:=261739; constant W46_2:integer:=131; constant
39 W47_2:integer:=476; constant B4_2:integer:=296;
40 constant W51_2:integer:=262003; constant W52_2:integer:=262096; constant
41 W53_2:integer:=262042; constant W54_2:integer:=142;
42 constant W55_2:integer:=261207; constant W56_2:integer:=164; constant
43 W57_2:integer:=805; constant B5_2:integer:=13;
44 constant W61_2:integer:=262114; constant W62_2:integer:=968; constant
45 W63_2:integer:=261244; constant W64_2:integer:=261832;
46 constant W65_2:integer:=250; constant W66_2:integer:=261884; constant
47 W67_2:integer:=261971; constant B6_2:integer:=261523
48 constant W71_2:integer:=663; constant W72_2:integer:=241; constant
49 W73_2:integer:=262113; constant W74_2:integer:=261462;
50 constant W75_2:integer:=239; constant W76_2:integer:=262050; constant
51 W77_2:integer:=262140; constant B7_2:integer:=330
52 constant W81_2:integer:=364; constant W82_2:integer:=54; constant
53 W83_2:integer:=261948; constant W84_2:integer:=213;
54 constant W85_2:integer:=35; constant W86_2:integer:=261696; constant
```

```
55 W87_2:integer:=261914; constant B8_2:integer:=260810;
56 constant W91_2:integer:=489; constant W92_2:integer:=261986; constant
57 W93_2:integer:=604; constant W94_2:integer:=262108;
58 constant W95_2:integer:=261810; constant W96_2:integer:=261687; constant
59 W97_2:integer:=401; constant B9_2:integer:=904;
60
61 constant W11_3:integer:=262086; constant W12_3:integer:=1374; constant
62 W13_3:integer:=262127; constant W14_3:integer:=262126;
63 constant W15_3:integer:=859; constant W16_3:integer:=262047; constant
64 W17_3:integer:=261678; constant W18_3:integer:=481;
65 constant W19_3:integer:=261771; constant B1_3:integer:=818;
66 constant W21_3:integer:=444; constant W22_3:integer:=261980; constant
67 W23_3:integer:=527; constant W24_3:integer:=136;
68 constant W25_3:integer:=6; constant W26_3:integer:=261611; constant
69 W27_3:integer:=78; constant W28_3:integer:=776;
70 constant W29_3:integer:=390; constant B2_3:integer:=261895);
71
72      port (tx: out std_logic;
73            clk, rst, we, push: in std_logic;
74            I, V: inout std_logic_vector (17 downto 0));
75     end Red_Neuronal_Pipeline;
76
77    architecture Arq of Red_Neuronal_Pipeline is
78
79    -----------SEÑALES PARA LA CONEXION DE LA RED NEURONAL---------------
80 signal cm_x1, cm_x2, cm_x3: std_logic_vector (17 downto 0);
81 signal c1_0,c1_1,c1_2,c1_3,c1_4,c1_5,c1_6,c1_7:std_logic_vector(17downto0);
82 signal c2_0,c2_1,c2_2,c2_3,c2_4,c2_5,c2_6,c2_7,c2_8,c2_9:std_logic_vector
83 (17downto0);
84 signal LOAD, Enable, clk_10, CE1, CE2, CE3: std_logic ;
85 signal SEL  : std_logic_vector (8 downto 0);
86 signal H, T: std_logic_vector (17 downto 0);
87 signal sal1, sal2: std_logic_vector (17 downto 0);
88 signal señal, señal2, em, etx, eload, stop: std_logic;
89
90    begin
91    LOAD <= SEL(8);
92    Enable <= SEL(7);
93    CE1 <= SEL(6);
94    CE2 <= SEL(5);
95    CE3 <= SEL(4);
96
97    em <= not Enable;

98    ------------------CONTROLADOR--------------------------
99    U0 : controlador port map (rst, clk, clk_10, SEL);
100
101   -----------------ENTRADA DE DATOS--------------------
102   Uin: Entrada port map (clk=>Enable, rst=>rst, H=>H, T=>T);
103
104   ------------------MULTIPLEXORES-----------------------
105   U1:MUX1_2x1 port map (H=>H, T=>T, SEL1=>SEL(0), Sal=>cm_x1);
106
107   U2:MUX2_8x1 port map (E0=>c1_0,E1=>c1_1,E2=>c1_2,E3=>c1_3,E4=>c1_4,
108   E5=>c1_5,E6=>c1_6,E7=>c1_7,SEL2=>SEL(2 downto 0),Sal=>cm_x2);
109
110   U3:MUX3_10x1 port map(E0=>c2_0,E1=>c2_1,E2=>c2_2,E3=>c2_3,E4=>c2_4,
```

```
111   E5=>c2_5,E6=>c2_6,E7=>c2_7,E8=>c2_8,E9=>c2_9,SEL3=>SEL(3 downto 0),
112   Sal=>cm_x3);
113
114   -----------------   CAPA1  -----------------------------
115   U4:Neurona_1
116   generic map(W1=>W11_1, W2=>W12_1, Bias=>B1_1)
117   port map (X1=>cm_x1, adr1=>SEL(0),LOAD=>LOAD,clk=>clk_10,Enable=>Enable,
118   CE=>CE1, sal1=>c1_0);
119   U5:Neurona_1
120
121   generic map(W1=>W21_1, W2=>W22_1, Bias=>B2_1)
122   port map (X1=>cm_x1,adr1=>SEL(0),LOAD=>LOAD,clk=>clk_10,Enable=>Enable,
123   CE=>CE1, sal1=>c1_1);
124
125   U6:Neurona_1
126   generic map(W1=>W31_1, W2=>W32_1, Bias=>B3_1)
127   port map (X1=>cm_x1,adr1=>SEL(0),LOAD=>LOAD,clk=>clk_10,Enable=>Enable,
128   CE=>CE1, sal1=>c1_2);
129
130   U7:Neurona_1
131   generic map(W1=>W41_1, W2=>W42_1, Bias=>B4_1)
132   port map (X1=>cm_x1,adr1=>SEL(0),LOAD=>LOAD,clk=>clk_10,Enable=>Enable,
133   CE=>CE1, sal1=>c1_3);
134
135   U8:Neurona_1
136   generic map(W1=>W51_1, W2=>W52_1, Bias=>B5_1)
137   port map (X1=>cm_x1,adr1=>SEL(0),LOAD=>LOAD,clk=>clk_10,Enable=>Enable,
138   CE=>CE1, sal1=>c1_4);
139
140   U9:Neurona_1
141   generic map(W1=>W61_1, W2=>W62_1, Bias=>B6_1)
142   port map (X1=>cm_x1,adr1=>SEL(0),LOAD=>LOAD,clk=>clk_10,Enable=>Enable,
143   CE=>CE1, sal1=>c1_5);
144   U10:Neurona_1
145   generic map(W1=>W71_1, W2=>W72_1, Bias=>B7_1)
146   port map (X1=>cm_x1,adr1=>SEL(0),LOAD=>LOAD,clk=>clk_10,Enable=>Enable,
147   CE=>CE1, sal1=>c1_6);
148
149   -----------------   CAPA2  -----------------------------
150   U11:Neurona_2
151   generic map(W1=>W11_2, W2=>W12_2, W3=>W13_2, W4=>W14_2, W5=>W15_2,
152   W6=>W16_2, W7=>W17_2, Bias=>B1_2)
153   port map (X2=>cm_x2, adr2=>SEL(2 downto 0), LOAD=>LOAD, clk=>clk_10,
154   Enable=>Enable, CE=>CE2, sal2=>c2_0);
155   U12:Neurona_2
156   generic map(W1=>W21_2, W2=>W22_2, W3=>W23_2, W4=>W24_2, W5=>W25_2,
157   W6=>W26_2, W7=>W27_2, Bias=>B2_2)
158   port map (X2=>cm_x2, adr2=>SEL(2 downto 0), LOAD=>LOAD, clk=>clk_10,
159   Enable=>Enable, CE=>CE2, sal2=>c2_1);
160
161   U13:Neurona_2
162   generic map(W1=>W31_2, W2=>W32_2, W3=>W33_2, W4=>W34_2, W5=>W35_2,
163   W6=>W36_2, W7=>W37_2, Bias=>B3_2)
164   port map (X2=>cm_x2, adr2=>SEL(2 downto 0), LOAD=>LOAD, clk=>clk_10,
165   Enable=>Enable, CE=>CE2, sal2=>c2_2);
166   U14:Neurona_2
167   generic map(W1=>W41_2, W2=>W42_2, W3=>W43_2, W4=>W44_2, W5=>W45_2,
```

```
168   W6=>W46_2, W7=>W47_2, Bias=>B4_2)
169   port map (X2=>cm_x2, adr2=>SEL(2 downto 0), LOAD=>LOAD, clk=>clk_10,
170   Enable=>Enable, CE=>CE2, sal2=>c2_3);
171
172   U15:Neurona_2
173   generic map(W1=>W51_2, W2=>W52_2, W3=>W53_2, W4=>W54_2, W5=>W55_2,
174   W6=>W56_2, W7=>W57_2, Bias=>B5_2)
175   port map (X2=>cm_x2, adr2=>SEL(2 downto 0), LOAD=>LOAD, clk=>clk_10,
176   Enable=>Enable, CE=>CE2, sal2=>c2_4);
177
178   U16:Neurona_2
179   generic map(W1=>W61_2, W2=>W62_2, W3=>W63_2, W4=>W64_2, W5=>W65_2,
180   W6=>W66_2, W7=>W67_2, Bias=>B6_2)
181   port map (X2=>cm_x2, adr2=>SEL(2 downto 0), LOAD=>LOAD, clk=>clk_10,
182   Enable=>Enable, CE=>CE2, sal2=>c2_5);
183
184   U17:Neurona_2
185   generic map(W1=>W71_2, W2=>W72_2, W3=>W73_2, W4=>W74_2, W5=>W75_2,
186   W6=>W76_2, W7=>W77_2, Bias=>B7_2)
187   port map (X2=>cm_x2, adr2=>SEL(2 downto 0), LOAD=>LOAD, clk=>clk_10,
188   Enable=>Enable, CE=>CE2, sal2=>c2_6);
189
190   U18:Neurona_2
191   generic map(W1=>W81_2, W2=>W82_2, W3=>W83_2, W4=>W84_2, W5=>W85_2,
192   W6=>W86_2, W7=>W87_2, Bias=>B8_2)
193   port map (X2=>cm_x2, adr2=>SEL(2 downto 0), LOAD=>LOAD, clk=>clk_10,
194   Enable=>Enable, CE=>CE2, sal2=>c2_7);
195
196   U19:Neurona_2
197   generic map(W1=>W91_2, W2=>W92_2, W3=>W93_2, W4=>W94_2, W5=>W95_2,
198   W6=>W96_2, W7=>W97_2, Bias=>B9_2)
199   port map (X2=>cm_x2, adr2=>SEL(2 downto 0), LOAD=>LOAD, clk=>clk_10,
200   Enable=>Enable, CE=>CE2, sal2=>c2_8);
201
202   ------------------   CAPA3  ------------------------------
203   U20:Neurona_3
204   generic map(W1=>W11_3, W2=>W12_3, W3=>W13_3, W4=>W14_3, W5=>W15_3,
205   W6=>W16_3, W7=>W17_3, W8=>W18_3, W9=>W19_3, Bias=>B1_3)
206   port map (X3=>cm_x3, adr3=>SEL(3 downto 0), LOAD=>LOAD, clk=>clk_10,
207   Enable=>Enable, CE=>CE3, sal3=>I);
208
209   U21:Neurona_3
210   generic map(W1=>W21_3, W2=>W22_3, W3=>W23_3, W4=>W24_3, W5=>W25_3,
211   W6=>W26_3, W7=>W27_3, W8=>W28_3, W9=>W29_3, Bias=>B2_3)
212   port map (X3=>cm_x3, adr3=>SEL(3 downto 0), LOAD=>LOAD, clk=>clk_10,
213   Enable=>Enable, CE=>CE3, sal3=>V);
214
215   ------------------SALIDA DE DATOS---------------------
216   Uout: Salida port map (I=>I,V=>V,tx=>tx,em=>em,enable=>Enable,we=>we,
217   clk_10=>clk_10, push=>push, rst=>rst);
218
219   end Arq;
```

9. Paquete creado para la Red Neuronal (Pack_Red.vhd)

```
1     library IEEE;
2     use IEEE.STD_LOGIC_1164.all;
3
4     package Pack_Red is
5
6     component Entrada is
7        port(clk,rst:  in    STD_LOGIC;
8                  H,T:   out STD_LOGIC_VECTOR( 17 downto 0));
9     end component;
10
11    component Salida is
12       port(tx: out std_logic;
13          I, V: in std_logic_vector (17 downto 0);
14          em, enable, we, clk_10, push, rst: in std_logic);
15
16    end component;
17
18    component Memoria_WR is
19        port(I, V: in std_logic_vector (17 downto 0);
20           señal2, señal, rst, push, we: in std_logic;
21           stop: inout std_logic;
22           Isal, Vsal: out std_logic_vector (17 downto 0));
23    end component;
24
25    component SERIAL is
26       port (I, V: in STD_LOGIC_VECTOR(17 downto 0);
27            eload, etx: inout STD_LOGIC;
28            clk, we, stop: in  STD_LOGIC;
29                        tx: out  STD_LOGIC);
30    end component;
31
32    component MUX is
33       port (em, eload, we: in std_logic;
34             señal: out std_logic);
35    end component;
36    component MUX2 is
37      port (Enable, etx, we: in std_logic;
38            señal2: out std_logic);
39    end component;
40
41    component clk_div is
42       port(CLK_IN1    : in  std_logic;
43            CLK_OUT1   : out std_logic;
44            CLK_OUT2   : out std_logic;
45            RESET      : in  std_logic;
46            LOCKED     : out std_logic);
47    end component;
48
49    component controlador is
50      port (rst, clk: in std_logic;
51              clk_out: out std_logic;
52                  sal: out std_logic_vector (8 downto 0));
53    end component;
54
```

```
55    component MUX1_2x1 is
56      port (H: in std_logic_vector (17 downto 0);
57            T: in std_logic_vector (17 downto 0);
58          Sal: out std_logic_vector (17 downto 0);
59         SEL1: in std_logic);
60    end component;
61
62    component MUX2_8x1 is
63     port (E0,E1,E2,E3,E4,E5,E6,E7: in std_logic_vector (17 downto 0);
64          SEL2: in std_logic_vector (2 downto 0);
65          Sal: out std_logic_vector (17 downto 0));
66    end component;
67
68    component MUX3_10x1 is
69  port (E0,E1,E2,E3,E4,E5,E6,E7,E8,E9: in std_logic_vector (17 downto 0);
70        SEL3: in std_logic_vector (3 downto 0);
71        Sal: out std_logic_vector (17 downto 0));
72  end component;
73
74    component ROM_1 is
75      generic (constant x1:integer:= 0;
76               constant x2:integer:= 0;
77               constant Size: Natural := 18);
78
79      port(adr1: in std_logic;
80           d: out std_logic_vector(size-1 downto 0));
81      end component;
82
83    component ROM_2 is
84      generic (constant x1:integer:= 0;
85               constant x2:integer:= 0;
86               constant x3:integer:= 0;
87               constant x4:integer:= 0;
88              constant x5:integer:= 0;
89               constant x6:integer:= 0;
90               constant x7:integer:= 0;
91               constant Size: Natural := 18);

92    port(adr2: in std_logic_vector (2 downto 0);
93         d: out std_logic_vector (size-1 downto 0));
94    end component;
95
96    component ROM_3 is
97      generic (constant x1:integer:= 0;
98               constant x2:integer:= 0;
99               constant x3:integer:= 0;
100              constant x4:integer:= 0;
101              constant x5:integer:= 0;
102              constant x6:integer:= 0;
103              constant x7:integer:= 0;
104              constant x8:integer:= 0;
105              constant x9:integer:= 0;
106             constant Size: Natural := 18);
107
108      port(adr3: in std_logic_vector (3 downto 0);
109           d: out std_logic_vector (size-1 downto 0));
110   end component;
```

```
111
112   component MAC is
113      Port (X, Y, B : in std_logic_vector(17 downto 0);
114            LOAD, CLK, CE : in std_logic;
115            A : out std_logic_vector(35 downto 0));
116   end component;
117
118   component TANSIG is
119      Port (X : in std_logic_vector(17 downto 0);
120            Y : out std_logic_vector(17 downto 0);
121            Enable : in std_logic);
122   end component;
123
124   component Neurona_1 is
125   generic (constant W1:integer :=0;
126            constant W2:integer :=0;
127            constant Bias: integer:=0);
128
129   Port (X1: in std_logic_vector(17 downto 0);
130         sal1: out std_logic_vector(17 downto 0);
131         LOAD, CLK, Enable, adr1, CE: in std_logic);
132
133   end component;
134
135   component Neurona_2 is
136   generic (constant W1:integer :=0;
137            constant W2:integer :=0;
138            constant W3:integer :=0;
139            constant W4:integer :=0;
140            constant W5:integer :=0;
141            constant W6:integer :=0;
142            constant W7:integer :=0;
143            constant Bias: integer:=0);
144
145   Port (X2: in std_logic_vector(17 downto 0);
146         sal2: out std_logic_vector(17 downto 0);
147         adr2: in std_logic_vector (2 downto 0);
148         LOAD, CLK, Enable, CE : in std_logic);
149   end component;
150
151   component Neurona_3 is
152   generic (constant W1:integer :=0;
153            constant W2:integer :=0;
154            constant W3:integer :=0;
155            constant W4:integer :=0;
156            constant W5:integer :=0;
157            constant W6:integer :=0;
158            constant W7:integer :=0;
159            constant W8:integer :=0;
160            constant W9:integer :=0;
161            constant Bias: integer:=0);
162
163    Port (X3: in std_logic_vector(17 downto 0);
164          sal3: out std_logic_vector(17 downto 0);
165          adr3: in std_logic_vector (3 downto 0);
166          LOAD, CLK, Enable, CE : in std_logic);
167
```

```
168    end component;
169
170   end Pack_Red;
```

Sección 6. Constantes.

1. Constantes de la Función de Activación.

```
%Constantes de la pendiente (a)
a0 = 0.9921875;      a1 = 0.923828125;
a2 = 0.80859375;     a3 = 0.66796875;
a4 = 0.52734375;     a5 = 0.400390625;
a6 = 0.296875;       a7 = 0.21484375;
a8 = 0.15234375;     a9 = 0.107421875;
aa = 0.07617875;     ab = 0.052734375;
ac = 0.037109375;    ad = 0.025390625;
ae = 0.025390625;    af = 0.01171875;

%Constantes de la ordenada en el origen (b)
b0 = 0;              b1 = 0.013671875;
b2 = 0.056640625;    b3 = 0.13671875;
b4 = 0.2421875;      b5 = 0.361328125;
b6 = 0.4765625;      b7 = 0.583984375;
b8 = 0.677734375;    b9 = 0.75390625;
ba = 0.8125;         bb = 0.861328125;
bc = 0.896484375;    bd = 0.923828125;
be = 0.921875;         bf = 0.958984375;

%Constantes del eje de las abscisas (x)
x0 = 0;             x1 = 0.1875;
x2 = 0.375;         x3 = 0.5625;
x4 = 0.75;          x5 = 0.9375;
x6 = 1.125;         x7 = 1.3125;
x8 = 1.5;           x9 = 1.6875;
xa = 1.875;         xb = 2.0625;
xc = 2.25;          xd = 2.4375;
xe = 2.625;         xf = 2.8125;
x10 = 3.5;
```