UNIVERSIDAD DE CONCEPCIÓN

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA ELÉCTRICA

Profesor Patrocinante:

Dr. Miguel E. Figueroa T.

Informe de Memoria de Título
para optar al título de:

Ingeniero Civil Electrónico

EMULADOR DE CIRCUITOS ADAPTIVOS ANÁLOGOS SOBRE UN FPGA

Concepción, Marzo de 2008

Daniel Esteban Herrera Peña

UNIVERSIDAD DE CONCEPCIÓN
Facultad de Ingeniería
Departamento de Ingeniería Eléctrica

Profesor Patrocinante:
Dr. Miguel E. Figueroa T.

EMULADOR DE CIRCUITOS ADAPTIVOS ANÁLOGOS SOBRE UN FPGA

Daniel Esteban Herrera Peña

Informe de Memoria de Título
para optar al Título de

Ingeniero Civil Electrónico

Marzo 2008

Resumen

Emulador de Circuitos Adaptivos Análogos sobre un FPGA

Daniel Esteban Herrera Peña
Universidad de Concepción, Marzo 2008

La implementación de algoritmos de procesamiento adaptivo de señales y aprendizaje de máquina en VLSI análogo y de señal mixta se dificulta por las características no ideales del hardware utilizado para diseñar las unidades funcionales de los algoritmos. Estas características incluyen: no linealidades, variación de parámetros, fuga de carga y dependencias con la temperatura en el circuito. Dado que el proceso de diseño e implementación de circuitos VLSI es largo y costoso, es necesario disponer de una herramienta que permita la evaluación y emulación del desempeño del algoritmo implementado antes de pasar a la etapa de fabricación. Simuladores de circuitos como SPICE permiten evaluar algunos de estos factores, pero, debido al bajo nivel al que estas herramientas simulan los circuitos, sus tiempos de ejecución son largos y no permiten la evaluación de sistemas de mediana y gran escala.

En esta Memoria de Título, se realiza el estudio, evaluación y análisis general de un algoritmo de aprendizaje y procesamiento adaptivo de señales, que presenta propiedades atractivas para ser implementado en VLSI. Se describe la operación y las principales características de algunas de las unidades aritméticas y de procesamiento utilizadas en VLSI análogo y de señal-mixta. Se estudia una herramienta software existente que simula estos circuitos. Y se desarrolla una herramienta hardware en Verilog HDL y Matlab®, que permite incorporar los efectos del hardware en la emulación de algoritmos, permitiendo la evaluación e interacción de sistemas de mediana y gran escala, facilitando el estudio de nuevas implementaciones y posibles mejoras en los circuitos para contrarrestar los efectos no deseados del hardware.

A toda mi familia que me ha brindado su apoyo incondicional, en especial a mis padres y Elena.

Agradecimientos

A todos mis compañeros que de algún u otro modo aportaron con el conocimiento logrado durante el desarrollo y preparación de esta memoria de título, al equipo de VLSI compuesto por Ricardo, Gonzalo y Waldo, quienes siempre dieron los mejor de sí cuando su ayuda solicité, a mi profesor patrocinante Dr. Miguel Figueroa quien me dió nuevos enfoques a los problemas que fueron apareciendo en este largo camino y a todos quienes utilizarán este informe para sus futuros trabajos e investigaciones.

Tabla de Contenidos

Lista de Tablas

Lista de Figuras

Nomenclatura

Matrices

R : matriz de autocorrelación.

Vectores

x : vector de universo de entrada.

w : vector de pesos.

$\hat{w}$: estimación de vector de pesos.

P : vector de correlación cruzada.

Escalares

e_k : error al instante k.

d_k : referencia al instante k.

y_k : salida red neuronal al instante k.

η : tasa de aprendizaje.

W : ancho del canal en transistor MOS.

D : constante de difusión de portadores.

N : densidad de portadores en transistor MOS.

z : distancia entre source y drain en transistor MOS.

$\phi(\cdot)$: función de activación.

$\chi(\cdot)$: número de condición.

Abreviaciones

Mayúsculas

CMOS	: Complementary Metal Oxide Semiconductor.
FPGA	: Field-Programmable Gate Array.
FNT	: Fowler-Nordheim Tunneling.
HDL	: Hardware Description Language.
ICA	: Independent Component Analysis.
IHEI	: Impact-ionized Hot Electron Injection.
MOSFET	: Metal Oxide Semiconductor Field-Effect Transistor.
MSE	: Mean Squared Error.
PCA	: Principal Component Analysis.
SOM	: Self-Organized Maps.
VLSI	: Very Large Scale Integration.

Minúsculas

d.c.	: direct current.

Capítulo 1. Introducción

En este capítulo se entregan una introducción al tema, se plantean los objetivos del estudio y se hace una sinopsis capítulo a capítulo del resto del trabajo.

1.1. Antecedentes Generales

Las sinapsis y neuronas en cerebros de animales codifican y procesan información usando señales químicas y eléctricas, con extraordinaria eficiencia, con una increíblemente pequeña potencia y restricciones de alimentación [14]. Estas observaciones han llevado a los investigadores a estudiar la biología como fuente de inspiración para diseño de ingeniería. También han provisto del impulso para la investigación en redes neuronales artificiales e ingeniería neuromorfológica.

Aunque las implementaciones contemporáneas de redes neuronales y algoritmos de aprendizaje de máquina, son basadas completamente en software, sigue siendo una perspectiva muy auspiciosa en funcionamiento, que los ingenieros pudieran construir versiones de hardware (silicio) de estas redes. Esta ideología se complica debido a la gran escala de unidades funcionales requerida para llevar a hardware un circuito adaptivo (en términos de costosas implementaciones en VLSI digital), pudiendo ésta solamente ser sintetizada a hardware mediante un diseño en VLSI análogo. El problema de diseñar circuitos análogos a gran escala radica en la alta no linealidad otorgada por el efecto *device mismatch*, el cual consiste en una variación imparcial de ganancias y offset en los bloques aritméticos, producidos por gradientes en los parámetros en el proceso de fabricación, los cuales crean variaciones en las propiedades físicas de los dispositivos de silicio a través de un chip, perjudicando dramáticamente el rendimiento general del sistema.

Con el pasar del tiempo, se ha logrado aliviar el efecto *device mismatch*, con lo cual se pueden diseñar hardware capaces de realizar los bloques aritméticos necesarios para la generación de un circuito adaptivo [2]. Además en [3] se ha diseñado un simulador de software para probar distintos algoritmos de procesamiento adaptivo de señales en VLSI que se apoya en la realización de bloques aritméticos con compensación para aminorar los efectos de hardware. Sin embargo, la naturaleza del software del simulador limita su desempeño y su capacidad de interactuar con circuitos digitales reales. La propuesta de este proyecto, es la elaboración de un emulador en hardware basado en FPGA con el propósito de verificar el funcionamiento de un circuito adaptivo, para su posterior fabricación en VLSI.

1.2. Trabajos Previos

El tema de la emulación de circuitos adaptivos no presenta antecedentes, por lo que se puede indicar que no existen trabajos previos. Sólo se cuenta con el simulador de algoritmos de procesamiento adaptivo de señales en VLSI [3], el cual forma una de las bases de esta memoria.

Como apoyo se encuentran los trabajos de redes adaptivas en [9] y [10], también se cuenta con la información acerca de las unidades aritméticas a utilizar de [2] y [8].

1.3. Hipótesis de Trabajo

El emulador de circuitos adaptivos análogos proveerá un paso intermedio entre el simulador software existente y la una posible solicitud de diseño en VLSI. Dejando la opción al usuario de verificar primero su diseño en software para luego emularla en tiempo real, comprobando el rendimiento esperado, las hipótesis más generales son:

- El emulador debe entregar resultados en tiempo menores a los entregados mediante simulación.
- La emulación en tiempo real permite la fácil integración con circuitería externa que requiera el desarrollo de algún algoritmo de procesamiento adaptivo.
- La herramienta debe ser capaz de construir esquemas de forma automática para simulación, síntesis e implementación en FPGA.

1.4. Objetivos

1.4.1. Objetivo General

El objetivo de esta memoria de título es desarrollar, diseñar e implementar un emulador basado en un FPGA con el fin de reproducir el comportamiento de algunos algoritmos de procesamiento adaptivo de señales, cuando son implementados en VLSI análogo y de señal mixta, mediante el desarrollo de una herramienta hardware que permita incorporar los modelos de las unidades aritméticas y de procesamiento con distintos niveles de abstracción, lo cual facilitará el estudio de nuevas implementaciones y permitirá evaluar posibles mejoras en los circuitos para contrarrestar los efectos no deseados del hardware, además este producirá resultados en tiempo real, lo que facilitará el acoplamiento con lógica que requiera el uso de un algoritmo implementado en

chip, posibilitando la interacción con el emulador y originará la decisión inmediata acerca de las posibilidades de implementarlo en VLSI.

1.4.2. Objetivos Específicos

- Estudiar y revisar el funcionamiento del simulador existente de circuitos adaptivos.
- Diseñar la arquitectura del simulador en hardware.
- Implementar en Verilog HDL el diseño para FPGA.
- Llevar a cabo simulaciones preliminares en Xilinx® ISE™ Webpack™.
- Diseñar interfaz de comunicación serial en Dev C++, para recepción de resultados.
- Diseñar script de Matlab®, para creación de circuito adaptivo configurable.
- Evaluar el emulador utilizando algoritmos de redes neuronales.

1.5. Alcances y Limitaciones

La herramienta hardware tendrá la capacidad de emular, basado en los bloques aritméticos, distintas configuraciones de algoritmos de procesamiento adaptivo de señales, teniendo este la capacidad de interactuar con otros circuitos para verificar el rendimiento del esquema bajo los efectos del hardware.

La tasa de aprendizaje se dejará fija en esta versión inicial del emulador, con el fin de darle más énfasis a la síntesis de los bloques aritméticos para en el futuro realizar perfecciones con respecto a este punto.

El ruido Gaussiano propio de un esquema conectado a un ambiente desconocido no será emulado en la herramienta a implementar.

1.6. Temario

El resto de este informe está estructurado de la siguiente forma:

- El Capítulo 2 entrega los fundamentos teóricos, principales aplicaciones y resultados experimentales del algoritmo LMS.
- El Capítulo 3 se encarga de detallar el diseño y funcionamiento de algunas de las unidades aritméticas y de procesamiento utilizadas en VLSI análogo y de Señal Mixta, se introduce además el simulador de circuitos adaptivo existente.

- El Capítulo 4 describe detalladamente, todos los pasos del diseño del emulador de circuitos adaptivos, revisando arquitecturas, diseño mismo e implementación en FPGA.
- El Capítulo 5 presenta las simulaciones de los sistemas implementados en VLSI análogo y de Señal Mixta.
- El Capítulo 6 muestra los resultados de las distintas emulaciones de los bloques constituidos en el Capítulo 4 y simulados en el Capítulo 5.
- El Capítulo 7 entrega las principales conclusiones del tema y bosqueja las direcciones del trabajo futuro.

Capítulo 2. Procesamiento Adaptivo de Señales y Redes Neuronales

2.1. Introducción

Las técnicas de procesamiento adaptivo de señales se basan en procesar los datos utilizando algoritmos parametrizados por un conjunto de coeficientes que representan las propiedades estadísticas de la entrada y/o del ambiente. La modificación de los coeficientes se realiza a través de un algoritmo de aprendizaje, el cual funciona en base a las entradas y salidas del sistema, y en algunos casos a una referencia externa. Las reglas que controlan la actualización de los coeficientes se derivan de criterios de optimización específicos para la tarea en cuestión.

Las redes neuronales artificiales son un campo atractivo para la implementación de estas técnicas, debido principalmente a la capacidad de procesamiento paralelo y robustez que presentan. Su estudio ha sido motivado, por el reconocimiento de que el cerebro humano es un sistema de procesamiento altamente complejo, no-lineal y paralelo. Y que tiene la capacidad de organizar las estructuras que lo conforman (neuronas), para realizar cálculos (Por Ejemplo; reconocimiento de patrones, percepción y control motriz), de forma completamente diferente y muchas veces más rápido que el computador más rápido en la actualidad. Como ejemplo, se puede mencionar la tarea rutinaria que realiza el cerebro al reconocer un rostro familiar, lo cual realiza en aproximadamente $100 - 200$ [ms], sin embargo, tareas mucho menos complejas pueden tomar días en un computador convencional.

De esta forma se puede decir que una red neuronal es una maquina diseñada para modelar la forma en la cual el cerebro realiza una tarea o función de interés. Las principales disciplinas en las que se encuentran presentes las redes neuronales son: neuro-ciencias, matemáticas, estadísticas, física, computación e ingeniería. Encontrando aplicaciones en diversos campos, tales como: reconocimiento de patrones, modelamiento, procesamiento de señales y control, entre otros.

2.2. Algoritmos de aprendizaje

El aprendizaje es un método mediante el cual los pesos de una red neuronal son adaptados. El tipo de aprendizaje está determinado por la forma en la cual se realiza la adaptación de los pesos. Existe una gran variedad de métodos de aprendizaje los que se pueden clasificar de varias maneras:

por su tipo de entrenamiento (supervisado o no supervisado), por el método de convergencia (gradiente descendente o minimización del error), por la función de activación de la neurona (lineal o no lineal), etc.

La elección de un algoritmo de aprendizaje en particular, está influenciada por la tarea que se quiera realizar y por las características de implementación de esta. Debido a que las implementaciones deben ser realizadas en VLSI, el algoritmo en cuestión debe poseer ciertas características especiales, las que se enumeran a continuación:

- Necesita comunicación local y estructurada.
- Usar aritmética simple, la cual lleve a implementaciones circuitales pequeñas y de baja potencia.
- Ser útil en diversas aplicaciones que requieran circuitos pequeños y de baja potencia.
- Robustez.

En las siguientes secciones se mencionarán los fundamentos teóricos y principales aplicaciones de algunos algoritmos de aprendizaje de redes neuronales atractivos de implementar en VLSI.

2.2.1. Algoritmo LMS

La red neuronal más simple, consiste de una única neurona con pesos sinápticos adaptables. El modelo de esta red se puede apreciar en la Fig. 2.1.

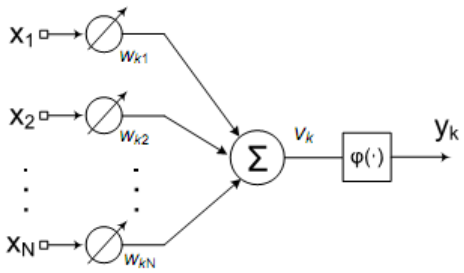


Fig. 2.1: Neurona con pesos sinápticos adaptables

Si la función $\phi(\cdot) = 1$ esta neurona forma la base de un filtro lineal adaptivo. Esta red también es conocida como perceptrón, siendo utilizada para clasificar patrones linealmente separables.

La salida de la red es una función $\phi(\cdot)$ de la combinación lineal v de la entrada x con los pesos sinápticos W, si la función de activación $\phi(\cdot)$ es la identidad, la red neuronal se denomina filtro lineal adaptivo. La modificación de los pesos del filtro lineal, se realiza a través del algoritmo LMS (least-mean-square), también conocido como regla delta, el que fue desarrollado por Widrow y Hoff en 1960.

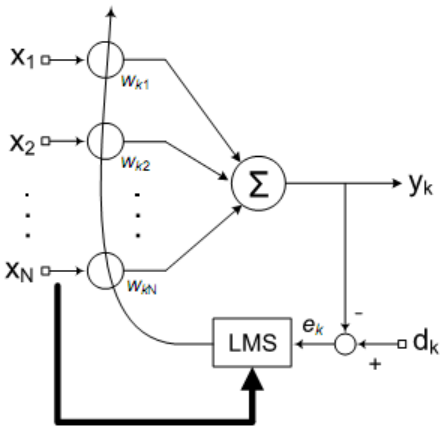


Fig. 2.2: Esquema del algoritmo de aprendizaje LMS para una neurona lineal

La modificación de los pesos sinápticos es función de la señal de error $e_k = d_k - y_k$ y de la entrada x_{ik}, i = 1, 2,. . ., N.

LMS corresponde a un algoritmo del tipo supervisado, debido al uso de una referencia que determina la señal de error, que se realimenta para modificar los pesos sinápticos de la red. A pesar del tiempo transcurrido desde su formulación, el algoritmo LMS es la herramienta por excelencia del filtrado adaptivo de señales, siendo además exitosamente aplicado en diversos campos, tales como: sistemas de comunicación, control, radar, sonar, antenas, sismología, e ingeniería biomédica, entre otros. Las principales virtudes que presenta el algoritmo son las siguientes:

- Formulación matemática sumamente sencilla.
- No requiere conocimiento de las propiedades del entorno
- Es lineal.
- Es robusto ante pequeñas perturbaciones e imprecisiones en el modelo, así como a algunas no idealidades aritméticas de los elementos computacionales.

Si se observa el filtro lineal adaptivo de la Fig. 2.2, se puede notar que este consta de dos procesos:

1. El proceso de filtrado, el cual involucra el cálculo de dos señales:
   - La salida y_k, que es producida en respuesta a las entradas, representadas por el vector x_k.
   - Una señal de error, e_k, que se obtiene comparando la salida y_k con la referencia d_k.
2. El proceso adaptivo, el cual involucra el ajuste automático de los pesos sinápticos de la neurona, en función de la señal de error e_k.

Las ecuaciones que describen estos procesos son las siguientes. La salida del filtro es la combinación lineal de la entrada y los pesos,

$$y_k = \sum_{i=1}^{N} w_{ik} x_{ik} \tag{2.1}$$

la que en forma matricial se expresa como un producto interior de vectores.

$$y_k = x_k^{\,T} w_k \tag{2.2}$$

donde los vectores w_k y x_k son de la forma.

$$w_k = \left[w_{1k}, w_{2k}, \cdots, w_{Nk} \right], \; x_k = \left[x_{1k}, x_{2k}, \cdots, x_{Nk} \right]$$

La señal de error se calcula comparando la referencia con la salida actual:

$$e_k = d_k - y_k \tag{2.3}$$

Esta señal de error es utilizada para controlar el ajuste de los pesos sinápticos de la neurona, el método mediante el cual se realiza este ajuste está determinado por la función de costo utilizada para derivar el algoritmo de aprendizaje de interés. En el caso de LMS, la función de costo busca

minimizar el error cuadrático medio (MSE), el cual se define como el valor esperado del cuadrado de la señal de error.

$$MSE = \varepsilon \triangleq \mathrm{E}\left[e_k^{\,2} \right] = \mathrm{E}\left[d_k^{\,2} \right] - 2E\left[d_k x_k^{\,T} \right] w + w^T E\left[x_k x_k^{\,T} \right] w \tag{2.4}$$

Para facilitar el análisis, se definen los términos R y P ([9]), como:

$$R \triangleq E\left[x_k x_k^{\,T} \right] \tag{2.5}$$

$$P \triangleq E\left[d_k x_k \right] = e \begin{bmatrix} d_k x_{1k} \\ d_k x_{2k} \\ \vdots \\ d_k x_{Nk} \end{bmatrix} \tag{2.6}$$

donde R es la matriz de autocorrelación de la entrada (simétrica y definida o semidefinida positiva) y P corresponde al vector de correlación cruzada entre la referencia y el vector de entrada. Ahora, al reescribir la ecuación (2.4) con los términos *R* y *P*, se tiene:

$$MSE = \mathrm{E}\left[d_k^{\,2} \right] - 2P^T w + w^T R\, w \tag{2.7}$$

De esta ecuación es posible observar que el MSE es una función cuadrática de los pesos, lo que en el caso de dos dimensiones forma un paraboloide con un mínimo global (Fig. 2.3). El proceso de adaptación de los pesos debe ser capaz de encontrar este mínimo de la función MSE.

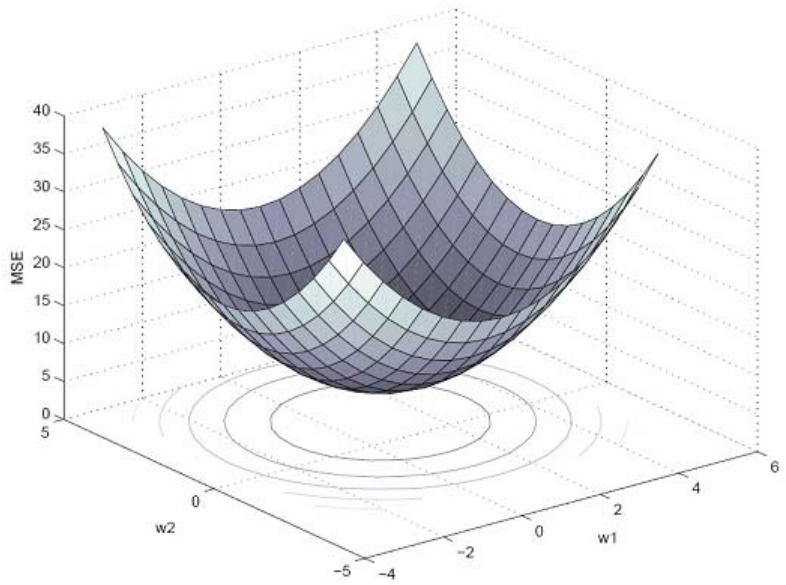


Fig. 2.3: Superficie de error MSE para el caso de dos dimensiones (dos pesos sinápticos)

El proceso de adaptación debe ser capaz de encontrar el mínimo de esta función a través de la modificación de los pesos.

2.2.2. Solución de Wiener y método Steepest Descent

El vector gradiente indica la dirección de máximo crecimiento de una función. Para la función del MSE, el gradiente está dado por la expresión:

$$\nabla \varepsilon(w) \triangleq \begin{bmatrix} \partial \varepsilon / \partial w_1 \\ \partial \varepsilon / \partial w_2 \\ \vdots \\ \partial \varepsilon / \partial w_N \end{bmatrix} = -2P + 2Rw \tag{2.8}$$

La solución que minimiza la función de error se encuentra cuando el gradiente es igual a cero, obteniendo el valor óptimo de los pesos w^*.

$$\nabla \varepsilon(w^*) = 0 \tag{2.9}$$

Resolviendo la ecuación (2.8), se obtiene que el valor óptimo de los pesos es:

$$w^* = R^{-1}P \tag{2.10}$$

Este valor óptimo es conocido como la solución de Wiener, sin embargo, para encontrar esta solución, se requiere conocer las estadísticas de segundo orden dadas por la matriz de correlación de la señal de entrada, (R) y por la correlación cruzada entre la señal de entrada y la referencia, (P). Lo cual no es posible en la mayoría de los casos prácticos.

Más allá de poder obtener la solución óptima, la solución de Wiener sirve como parámetro de referencia del algoritmo LMS. Se define el valor mínimo del MSE [10], el cual se obtiene cuando el vector de pesos w alcanza su valor óptimo w^*.

$$\varepsilon_{\min} = E\left[d_k{}^2\right] - P^T w^* \tag{2.11}$$

Este valor permite redefinir el MSE de la siguiente forma:

$$\varepsilon = \varepsilon_{\min} + v^T R\, v \tag{2.12}$$

donde el vector v se define como la diferencia entre los vectores w y w^*.

$$v \triangleq w - w^*$$

Además, como la matriz R es simétrica y definida positiva, se puede descomponer en la forma:

$$R = Q \Lambda Q^T$$

donde Q es una matriz que tiene como columnas a los vectores propios y Λ es una matriz diagonal conteniendo los valores propios de R.

A través de esta descomposición, es posible definir una transformación v', tal que la definición para el MSE pueda ser expresada de una forma más clara.

$$v' \triangleq Q^T v$$

Así la nueva definición para el MSE es:

$$\varepsilon = \varepsilon_{\min} + v'^{T} \Lambda v' \tag{2.13}$$

El método steepest descent, utiliza la dirección opuesta del vector gradiente para encontrar el mínimo de la función de costo, modificando el vector de pesos w de la siguiente manera:

$$w_{k+1} = w_k - \eta \nabla \varepsilon(w)_k \tag{2.14}$$

donde η es una constante positiva llamada tasa de aprendizaje y $\nabla \varepsilon(w)_k$ es el vector gradiente evaluado en el punto w_k. El método de steepest descent converge a la solución óptima de Wiener w^* lentamente, teniendo la tasa de aprendizaje η una profunda influencia en su comportamiento:

- Cuando η es pequeño, la respuesta transiente del algoritmo es sobre-amortiguada, y la trayectoria trazada por w_k sigue un curso suave.
- Cuando η es grande, la respuesta transiente del algoritmo es sub-amortiguada, en este caso la trayectoria seguida por w_k sigue una ruta zigzagueante (oscilatoria).
- Cuando η excede un cierto valor crítico, el algoritmo se vuelve inestable (diverge).

2.2.3. Formulación del Algoritmo LMS

El algoritmo LMS está basado en el uso de valores instantáneos para la función de costo [11]:

$$\varepsilon(w) = \frac{1}{2} e_k^{\,2} \tag{2.15}$$

Derivando $\varepsilon(w)$ con respecto al vector de pesos w, se obtiene:

$$\frac{\partial \varepsilon(w)}{\partial w} = e_k \frac{\partial e_k}{\partial w} \tag{2.16}$$

Reemplazando 2.2 en 2.3, es posible expresar la señal de error como;

$$e_k = d_k - x_k^T w_k \tag{2.17}$$

de esta forma, se obtienen las expresiones:

$$\frac{\partial e_k}{\partial w} = -x_k$$

$$\frac{\partial \varepsilon(w)}{\partial w} = -x_k e_k \tag{2.18}$$

Utilizando este último resultado como una estimación del vector gradiente, se puede escribir:

$$\hat{\nabla}\varepsilon(w)_k = -x_k e_k \tag{2.19}$$

Luego, utilizando la ecuación (2.19) para el vector gradiente en la ecuación (2.14), se puede formular el algoritmo LMS como sigue:

$$\hat{w}_{k+1} = \hat{w}_k + \eta\, x_k\, e_k \tag{2.20}$$

En la ecuación (2.20) se ha utilizado el término $\hat{w}_k$ en lugar de w_k para recalcar el hecho de que el algoritmo LMS produce una estimación del vector de pesos que resultaría del uso del método steepest descent. A diferencia de lo que sucede con steepest descent, el cual presenta trayectorias bien definidas en la convergencia de los pesos, en el algoritmo LMS el vector de pesos $\hat{w}_k$ sigue trayectorias aleatorias. Y a medida que el número de iteraciones se aproxima a infinito, $\hat{w}_k$ se mueve aleatoriamente alrededor de la solución de Wiener.

Un hecho de suma importancia es que a diferencia del método de steepest descent y de la solución de Wiener, el algoritmo LMS *no requiere conocimiento sobre las estadísticas del ambiente.*

2.2.4. Convergencia de LMS

El comportamiento del algoritmo LMS está regido por dos variables diferentes; la tasa de

aprendizaje η y el vector de entrada x_k. Es así como la estabilidad del algoritmo está influenciada por las características estadísticas del vector de entrada y por el valor de la tasa de aprendizaje η. Debido a que las estadísticas de la entrada son desconocidas y no pueden ser alteradas, la convergencia del algoritmo queda completamente regida por la elección del valor de la tasa de aprendizaje η.

El criterio de convergencia utilizado para el análisis del algoritmo LMS, es la convergencia en la media cuadrática, descrita por:

$$E\left[e_k^{\,2}\right] \rightarrow constante\, como\, n \rightarrow \infty \tag{2.21}$$

Para realizar este análisis, sin embargo, se deben realizar una serie de suposiciones [11]:

1. Los sucesivos vectores de entrada son estadísticamente independientes entre ellos.
2. A la iteración *k*, el vector de entrada x_k es estadísticamente independiente de todas la muestras previas de la respuesta deseada, d_1, d_2, . . . , d_{k-1}.
3. A la iteración *k*, la respuesta deseada d_k es dependiente de x_k, pero estadísticamente independiente de todas las respuestas deseadas previas.
4. El vector de entrada x_k y la respuesta deseada d_k están dadas por distribuciones Gaussianas.

El resultado del análisis, asumiendo una tasa de aprendizaje muy pequeña, muestra que el algoritmo LMS converge en la media cuadrática, siempre que η cumpla la condición:

$$0 < \eta < \frac{2}{\lambda_{\max}} \tag{2.22}$$

donde $\lambda_{\max}$ es el valor propio más grande de la matriz de correlación de entrada *R*. Sin embargo, en aplicaciones típicas de LMS, no es posible conocer $\lambda_{\max}$. Una forma alternativa consiste en utilizar una estimación conservativa de $\lambda_{\max}$ dada por la traza de *R*, en tal caso la condición dada por la ecuación (2.22), toma la siguiente forma.

$$0 < \eta < \frac{2}{tr[R]} \tag{2.23}$$

donde $tr[R]$ corresponde a la traza de la matriz *R*.

Como se mencionó en el comienzo de la sección, LMS posee una gran cantidad de virtudes, las que lo han hecho un algoritmo sumamente popular durante mucho tiempo, sin embargo, también presenta limitaciones. Las principales son: la lenta tasa de convergencia y la sensibilidad a variaciones en las condiciones del ambiente. El problema de la convergencia lenta, se ve particularmente empeorado cuando la dimensión de la entrada es alta. Y el algoritmo es especialmente sensible a variaciones en el número de condición o spread de los valores propios de la matriz de correlación *R*, definido por el cuociente entre el valor propio más grande y el valor propio más pequeño:

$$\chi(R) = \frac{\lambda_{\max}}{\lambda_{\min}} \tag{2.24}$$

En general, mientras mayor sea la correlación entre las componentes del vector de entrada x_k, mayor será la dispersión de los valores propios. Si las componentes de x_k son completamente ortogonales, la dispersión es 1 y LMS tiene un desempeño óptimo. Esta dependencia de las estadísticas de la entrada es la mayor desventaja de LMS. Algoritmos como Mínimos Cuadrados Recursivos (RLS) mejoran su desempeño manteniendo una estimación de la inversa de *R* para desacoplarse de las estadísticas de la entrada, pero sacrifican la simplicidad, regularidad y localidad de LMS.

Una forma útil de medir el comportamiento del algoritmo LMS es a través del desajuste (misadjustment), *M*, el cual está dado por:

$$M = \frac{\varepsilon_\infty - \varepsilon_{\min}}{\varepsilon_{\min}} \tag{2.25}$$

donde $\varepsilon_{\min}$ está dado por la ecuación (2.11) y ε_∞ corresponde al valor de la función de costo en

estado estable. El desajuste *M* es directamente proporcional a la tasa de aprendizaje η, y la constante de tiempo de convergencia es inversamente proporcional a esta. O sea, tasas de aprendizaje altas convergen más rápido, pero generan un mayor desajuste. Existe por lo tanto, un compromiso entre desajuste y velocidad de convergencia, que hace crucial una buena elección de la tasa de aprendizaje para producir una respuesta global satisfactoria.

Una modificación en la tasa de aprendizaje, que podría mejorar el desempeño de LMS, es hacerla variable en el tiempo. La idea es partir con una tasa de aprendizaje alta para acelerar la convergencia, pero reducirla en el tiempo para minimizar el desajuste del error. Una expresión comúnmente utilizada es.

$$\eta_k = \frac{c}{k} \tag{2.26}$$

Aquí la tasa se reduce rápidamente, pero *c* no puede ser muy grande. Otra alternativa está dada por el método llamado búsqueda y convergencia,

$$\eta_k = \frac{\eta_0}{1+(k/\tau)} \tag{2.27}$$

donde η_0 y τ son seleccionadas por el usuario. En las etapas iniciales del entrenamiento, para valores de *k* pequeños en comparación con τ , la tasa de aprendizaje η_k es aproximadamente igual a η_0, y el algoritmo opera esencialmente como LMS estándar. Luego para un valor de *k* grande en comparación con τ, la tasa de aprendizaje η_k se aproxima a *c*/*k*, con $c = \tau\eta_0$, de esta forma los pesos convergen a sus valores óptimos.

2.2.5. Pruebas Experimentales

Se realizaron una gran cantidad de pruebas para comprobar el funcionamiento del algoritmo LMS en la solución de algunos problemas clásicos, entre los que se encuentran: la clasificación de patrones linealmente separables, como es el caso de las compuertas lógicas OR y AND descritos en

[11], [9]; el reconocimiento de dígitos; y la implementación de un filtro lineal adaptivo. Esta última aplicación es la que detallaremos a continuación.

Se implementó un filtro lineal adaptivo de 10 entradas (Fig. 2.4) entrenado con el algoritmo LMS. La entradas son variables aleatorias de una distribución uniforme entre [−1, 1], la señal de referencia es obtenida presentando la misma secuencia de entrada a una neurona con pesos fijos (W^*) más un ruido de medición Gaussiano de media cero y varianza $\sigma^2 = 10^{-6}$. El valor de la tasa de aprendizaje se mantiene constante durante todo el proceso de aprendizaje.

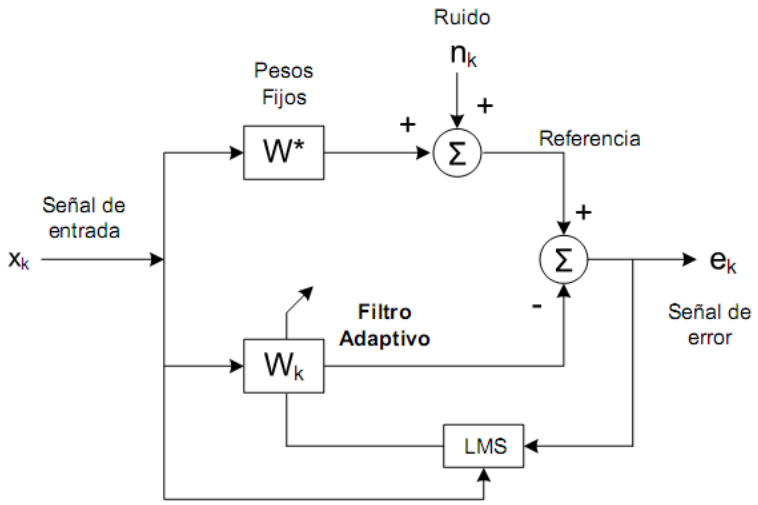


Fig. 2.4: Filtro lineal adaptivo entrenado con el algoritmo LMS

La primera prueba consistió en entrenar el filtro utilizando diferentes tasas de aprendizaje para observar el efecto de este valor en la velocidad de convergencia y error residual. En la Fig. 2.5, se observa la evolución del MSE (graficado en ventana deslizante de 100 puntos), para cuatro diferentes valores de la tasa de aprendizaje. Además, en la figura 2.6 se muestra la evolución del peso 1 y 8 de la red para las distintas tasas de aprendizaje utilizadas.

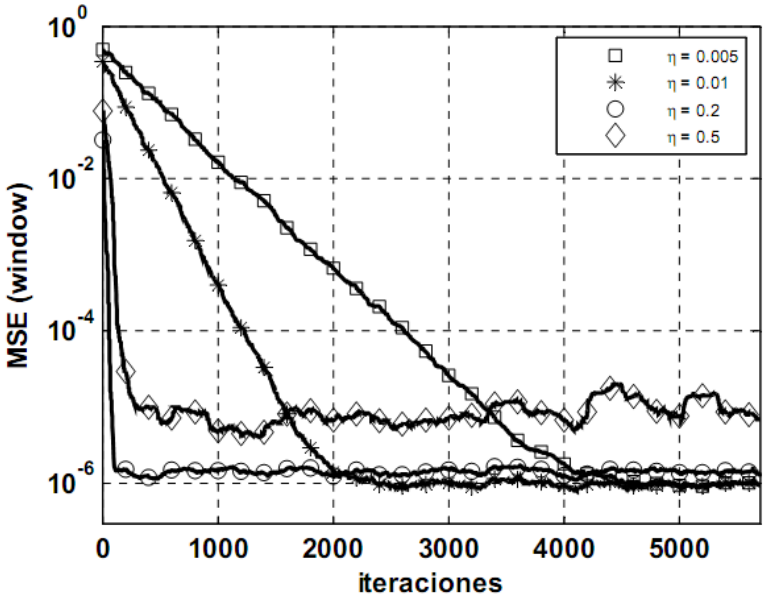
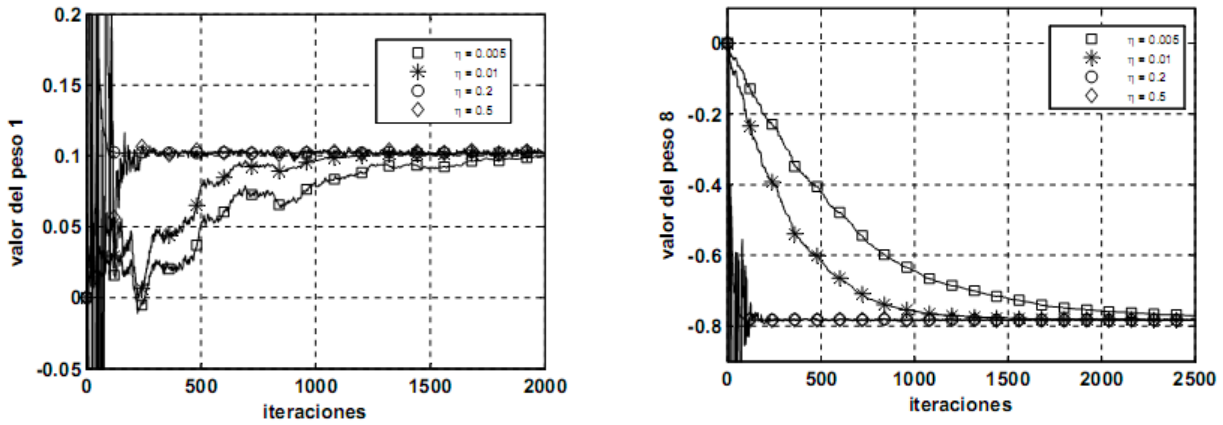


Fig. 2.5: Evolución del MSE para un filtro adaptivo entrenado con el algoritmo LMS

Se observa la diferencia en la velocidad de convergencia y error residual para cuatro tasas de aprendizaje diferentes. El error mínimo está dado por la varianza del ruido de medición (σ^2= 10−^{6}). Una tasa de aprendizaje pequeña presenta un menor error residual y menor velocidad de convergencia. En el caso de una tasa de aprendizaje alta, la velocidad de convergencia es mayor pero el MSE presenta una gran varianza y un mayor error residual. El valor de la tasa de aprendizaje debe ser elegido de acuerdo a un compromiso entre velocidad de convergencia y error residual.

En la Fig. 2.5 se aprecia claramente el efecto del valor de la tasa de aprendizaje en la velocidad de convergencia y error residual. Una tasa de aprendizaje baja permite obtener un error residual y varianza pequeños, sin embargo, la velocidad de convergencia se hace muy lenta. De forma contraria una tasa de aprendizaje alta aumenta la velocidad de convergencia pero con el costo de un mayor error residual y mayor varianza. De esta forma, la elección de la tasa de aprendizaje debe ser realizada teniendo en cuenta el compromiso entre velocidad de convergencia y error residual. En la Fig. 2.6, además se puede observar la evolución de dos de los pesos de la red, aquí también es claro el efecto del valor de la tasa de aprendizaje.



(a) Evolución del peso 1 (b) Evolución del peso 8

Fig. 2.6: Evolución del filtro adaptivo entrenado con LMS y usando diferente tasas de aprendizaje

(**a**) Peso 1. (**b**) Peso 8. La convergencia de los pesos 1 y 8 presentan constantes de tiempo diferentes, sin embargo, al igual que en el caso del MSE, una tasa de aprendizaje baja lleva a tiempos de convergencia mayores y pesos finales con menor varianza. En cambio con tasas de aprendizaje altas los pesos presentan una mayor varianza.

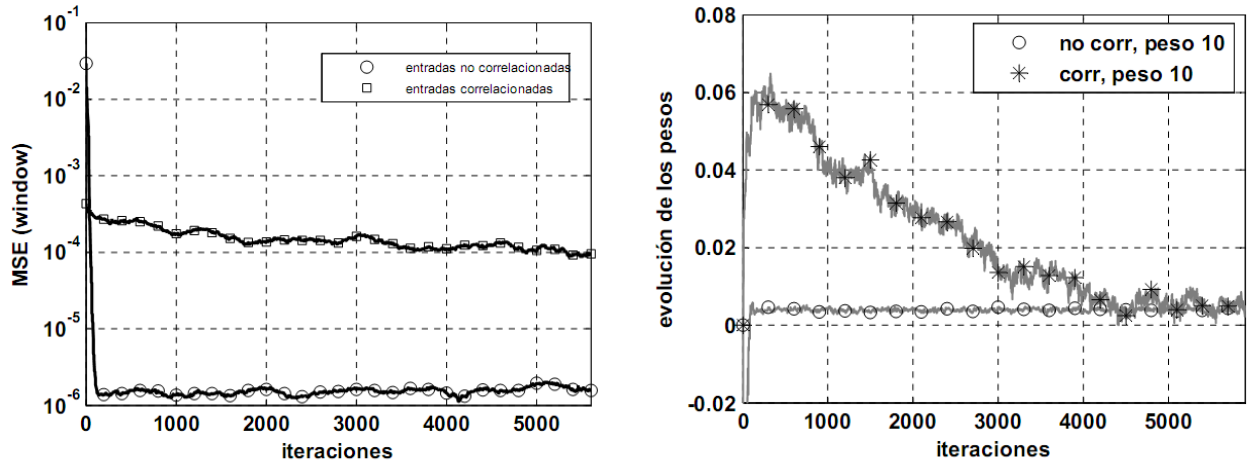
Como se mencionó anteriormente, la mayor desventaja que presenta el algoritmo LMS es su dependencia de las estadísticas de la entrada. Por esto, la segunda prueba, consistió en comparar el desempeño de LMS ante entradas no correlacionadas y correlacionadas. El primer conjunto de entrada es el mismo utilizado en la prueba anterior, el segundo conjunto de entrada se obtuvo de multiplicar el primer conjunto por una matriz de mezcla M que imita el funcionamiento de un arreglo de antenas equidistantes, donde cada entrada representa un sensor del arreglo que recibe una combinación lineal de todas las señales. Las entradas correlacionadas tienen la siguiente forma.

$$
\begin{aligned}
x_{1c} &= x_1 + \ldots + 0.69 \cdot x_5 + \ldots + 0.3 \cdot x_{10} \\
\vdots &= \vdots \\
x_{5c} &= 0.69 \cdot x_1 + \ldots + x_5 + \ldots + 0.61 \cdot x_{10} \\
\vdots &= \vdots \\
x_{10c} &= 0.3 \cdot x_1 + \ldots + 0.61 \cdot x_5 + \ldots + x_{10}
\end{aligned}
$$

Para ambos casos se utilizó el mismo vector de referencia y pesos iniciales, la tasa de aprendizaje corresponde al mejor valor encontrado para cada caso, $\eta = 0.2$ para entradas no correlacionadas y $\eta = 0.04$ para entradas correlacionadas. El valor de la tasa de aprendizaje representa la primera diferencia entre ambos casos, pues el valor que permite hacer estable al

algoritmo con entradas correlacionadas es bastante menor que el encontrado para el otro caso.

En la Fig. 2.7 se observa la evolución del MSE y de los pesos asociados a la entrada 10 en las dos situaciones planteadas. Debido al acoplamiento entre las entradas, la modificación de un peso en la red afecta a todos los demás, por lo que el desempeño del algoritmo es bastante malo en comparación con el caso de entradas no correlacionadas. En la sección 2.4.2.1 se verá una red que permite la decorrelación de señales y que permite una aceleración en la convergencia y disminución del error residual para el algoritmo LMS.



(a) Evolución del MSE (b) Evolución del peso 10

Fig. 2.7: Evolución del filtro lineal adaptivo ante entradas correlacionadas y no correlacionadas

(**a**) MSE para ambos casos. Se puede apreciar claramente como el desempeño del algoritmo LMS se reduce considerablemente al operar con entradas correlacionadas, presentando tiempos de convergencia órdenes de magnitud más grandes que en el caso ideal y errores residuales eventualmente mayores. (**b**) Evolución del peso asociado a la entrada 10 del filtro adaptivo. La velocidad de convergencia se reduce considerablemente y la varianza del peso es alta debido al acoplamiento entre las entradas, la modificación de un peso en la red afecta a todos los demás.

Capítulo 3. Hardware y Simulador de Algoritmos de Aprendizaje en VLSI

En este capítulo se establecen las bases teóricas de operación de algunas de las unidades aritméticas y de procesamiento utilizadas en implementaciones de redes neuronales en VLSI análogo, además se introduce el simulador de circuitos adaptivos existente.

3.1. Introducción

El algoritmo de aprendizaje descrito en el capítulo anterior demostró ser útil en diversas aplicaciones de procesamiento de señales, tales como: filtrado adaptivo, decorrelación de señales y compresión de datos. Se derivan además de criterios de optimización similares y pueden ser implementados en VLSI utilizando un set común de arquitecturas y primitivas computacionales.

La Fig. 3.1 muestra un esquema general de un sistema de aprendizaje típico. El forward path implementa un mapeo entre el espacio de entrada y el de salida a partir de los pesos sinápticos que representan las estadísticas de la entrada. Esto se realiza generalmente a través de la multiplicación del vector de entrada por el vector o matriz de pesos sinápticos, por lo cual las operaciones involucradas en este paso son un grupo de multiplicaciones, sumas y a veces una función no-lineal. En la sección 3.3 se realiza una descripción de los principios de operación de un multiplicador análogo de Gilbert operando bajo-umbral.

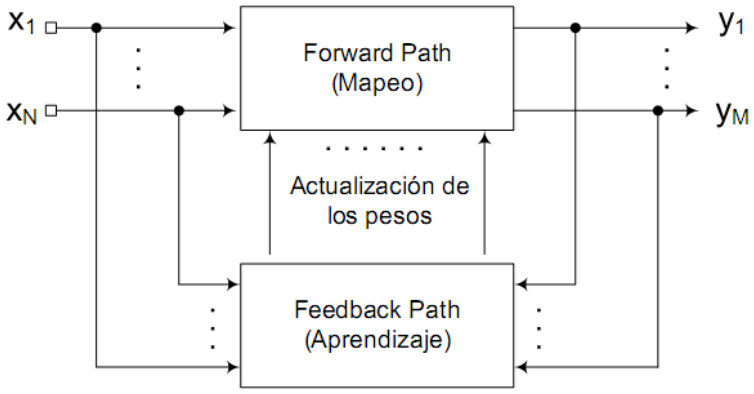


Fig. 3.1: Sistema general de aprendizaje

El forward path realiza un mapeo entrada-salida basado en los pesos sinápticos de la red. El feedback path se encarga del almacenamiento y de la actualización de los pesos basándose en el valor de las entradas y de una señal de realimentación.

El feedback path almacena los pesos sinápticos y actualiza sus valores basado en la señales de entrada y de realimentación. Como vimos en el capítulo anterior, la actualización se realiza la mayoría de las veces a través de la multiplicación entre algunas de estas señales, por lo que existe una similitud entre el cálculo realizado en el forward y en el feedback path. Sin embargo, la diferencia principal radica en que la operación de adaptación en el feedback path se realiza a una velocidad menor a la de cálculo en el forward path.

Los dispositivos utilizados para almacenar los pesos han sido clásicamente capacitores para valores análogos y registros para valores digitales. En la sección 3.4 se describe el funcionamiento de algunos de estos dispositivos y se hace énfasis en un método no volátil para almacenar y actualizar un valor análogo.

3.2. Sumadores

La suma de corrientes en circuitos análogos es particularmente elegante. La ley de corrientes de Kirchhoff, establece que la suma de corrientes en un nodo es cero, y de esta forma un sumador de corrientes es simplemente la unión de un cable con otro. De esta forma, la implementación de una función que realice la suma de dos corrientes se realiza sin mayores inconvenientes.

3.3. Multiplicadores Análogos

Debido a que los multiplicadores digitales ocupan gran cantidad de espacio y presentan un alto consumo de energía, las implementaciones de filtros adaptivos y redes neuronales con estos dispositivos rara vez son llevadas a cabo, y están usualmente limitadas a sistemas de pequeña envergadura. Para sistemas medianamente grandes son utilizados multiplicadores análogos, los cuales son bloques funcionales fundamentales en un gran número de aplicaciones de procesamiento como duplicador de frecuencia y modulador.

Los multiplicadores análogos son a menudo clasificados de acuerdo al rango que presentan sus entradas, como multiplicadores de: *1 cuadrante* (unipolar), *2 cuadrantes* (donde sólo una entrada puede ser bipolar), o *4 cuadrantes* (donde ambas entradas pueden ser bipolares).

Se puede encontrar en la literatura una gran cantidad de diseños de multiplicadores análogos en tecnología CMOS, que utilizan las características de los transistores operando en la región sobre-umbral, presentando rangos lineales de hasta unos pocos volts. Sin embargo, debido a las restricciones de consumo de energía que impone el diseño de aplicaciones portables, estos dispositivos no son factibles de ser utilizados. Para estos casos es necesario que los bloques funcionales operen los transistores en la región bajo-umbral, lo cual tiene la ventaja de que los niveles de corriente son típicamente órdenes de magnitud inferiores que en los dispositivos polarizados sobre-umbral y digitales.

3.3.1. Fundamentos de Operación

Para entender los principios de operación del multiplicador análogo operando bajo-umbral se mostrará un análisis de los bloques que lo componen. En primer lugar, revisando las características de funcionamiento del transistor MOS operando bajo-umbral, luego las características que presenta el transistor conectado en forma de diodo, para finalizar con los espejos de corriente y par diferencial.

3.3.1.1. Transistor operando bajo umbral

Para entender los principios de operación del multiplicador análogo operando bajo-umbral se mostrará un análisis de los bloques que lo componen. En primer lugar, revisando las características de funcionamiento del transistor MOS operando bajo-umbral, luego las características que presenta el transistor conectado en forma de diodo, para finalizar con los espejos de corriente y par

diferencial.

En un transistor MOS, la cantidad de corriente que fluye desde source a drain es controlada por el campo eléctrico generado por un voltaje aplicado al gate. El campo eléctrico atrae portadores de carga desde ambos lados del canal al source y drain, formando una delgada capa conductiva entre ellos. Un mayor voltaje y un campo eléctrico más grande se traducen en un mayor flujo de corriente a través del transistor. En la figura 4.1 se muestra el corte transversal y el símbolo de un transistor NMOS, además del símbolo y las definiciones para un transistor PMOS.

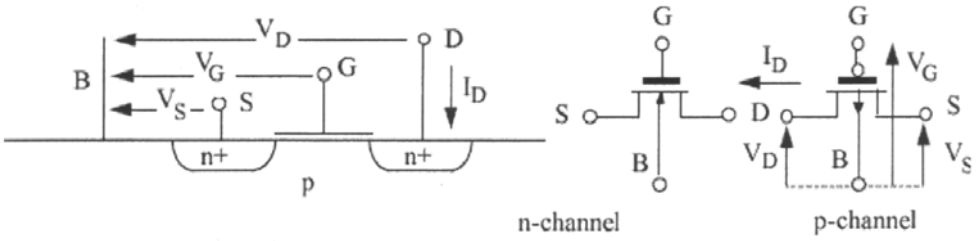


Fig. 3.2: Corte transversal esquemático de un transistor NMOS y símbolos asociados

Todos los voltajes están referidos al substrato local. Los voltajes y corrientes positivas para el transistor PMOS han sido invertidos para mantener la validez del modelo derivado para el NMOS.

En un semiconductor, existen dos modos por los cuales la corriente puede fluir: difusión y drift. La *difusión* es el flujo natural de partículas desde altas concentraciones hacia bajas concentraciones, y el *drift* es el flujo de partículas sujeto a una fuerza aplicada. En la región bajo-umbral de un transistor MOS, el flujo de corriente desde source a drain es debido a difusión. La corriente de difusión en un transistor MOS está dada por la ecuación,

$$I = -WqD\frac{\partial N}{\partial z} \tag{3.1}$$

donde W es el ancho del canal, D es la constante de difusión de los portadores, N es su densidad y z es la distancia entre source y drain. La densidad de los portadores decrece linealmente a lo largo del canal, así $\partial N/\partial z$ puede ser simplificado a $N_d - N_s/L$. Donde N_s es la densidad de portadores en el source, y N_d es la densidad en el drain:

$$N_s = N_1 e^{\frac{-q\psi_s}{kT}} e^{\frac{qV_s}{kT}} \tag{3.2}$$

$$N_d = N_1 e^{\frac{-q\psi_s}{kT}} e^{\frac{qV_d}{kT}} \tag{3.3}$$

De esta forma, reemplazando N_d y N_s por sus respectivas fórmulas en la ecuación:

$$I = -WqD\frac{N_d - N_s}{L}$$

Se obtiene:

$$I = \frac{qW}{L} N_1 D e^{\frac{-q\psi_s}{kT}} \left(e^{\frac{qV_s}{kT}} - e^{\frac{qV_d}{kT}} \right) \tag{3.4}$$

donde ψ_s es la superficie potencial en el source y a lo largo del canal.

Si se asume que las excursiones alrededor del punto de operación son pequeñas, es posible reemplazar ψ_s con κV_g. Luego reuniendo todas las constantes pre-exponenciales en un solo término, I_o, se tiene:

$$I = I_0 e^{\frac{-q\kappa V_g}{kT}} \left(e^{\frac{qV_s}{kT}} - e^{\frac{qV_d}{kT}} \right) \tag{3.5}$$

Esta ecuación es la que describe la operación bajo-umbral del transistor PMOS. En un transistor NMOS, un incremento de voltaje de gate atrae portadores de carga, de esta forma los voltajes se invierten:

$$I = I_0 e^{\frac{q\kappa V_g}{kT}} \left(e^{\frac{-qV_s}{kT}} - e^{\frac{-qV_d}{kT}} \right) \tag{3.6}$$

Para simplificar esta ecuación, se asume que el substrato es intrínseco, de esta manera el término *k* puede ser excluido. Además, debido a que *kT/q* es un voltaje que solo cambia con la

temperatura, puede reemplazarse por una única variable, V_T :

$$I = I_0 e^{\frac{V_g}{V_T}} \left(e^{\frac{-V_s}{V_T}} - e^{\frac{-V_d}{V_T}} \right) \tag{3.7}$$

Otra forma de esta ecuación, menos exacta, pero más fácil de manipular algebraicamente, asume que el transistor está en saturación, donde cambios de V_d no afectan fuertemente sobre el flujo de corriente a través del transistor. Esta región comienza cuando $|V_d - V_s| \geq 4kT/q \approx 100mV$. En este modo, el término V_d se elimina, quedando:

$$I = I_0 e^{\frac{V_g}{V_T}} e^{\frac{-V_s}{V_T}} = I_0 e^{\frac{V_g - V_s}{V_T}} \tag{3.8}$$

A partir de esta ecuación, es posible establecer un modelo para el transistor con una simple respuesta exponencial.

3.3.1.2. Transistor operando como diodo

Un transistor operando como diodo (Fig. 3.3), permite obtener una respuesta logarítmica muy útil en un sin número de implementaciones. Resolviendo la ecuación (3.9) para V_g , es posible observar que un transistor CMOS saturado y bajo-umbral operando como diodo toma el logaritmo de su corriente de entrada:

$$V_g = V_T \ln\left(\frac{I}{I_0}\right) + V_s \tag{3.9}$$

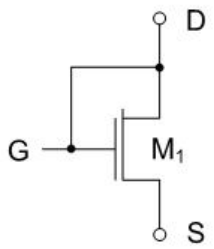


Fig. 3.3: Transistor operando como diodo

Para procesos típicos, I_0 es lo suficientemente grande que aún para corrientes de drain muy pequeñas, el voltaje en el gate del transistor estará alrededor de los 0.4 [V]. Esto asegura que un transistor operando como diodo estará siempre saturado y la ecuación anterior será válida para todos los valores bajo-umbral útiles.

3.3.1.3. Espejos de corriente

Una extensión natural del transistor operando como diodo es un espejo de corriente (Fig. 3.4), donde el gate de otro transistor es conectado a la entrada del transistor operando como diodo.

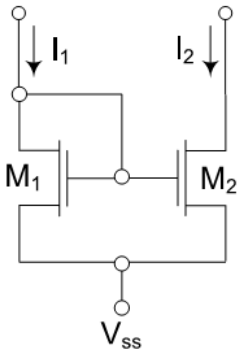


Fig. 3.4: Espejo de corriente utilizando transistores NMOS

El transistor M2 crea una copia de la corriente fluyendo a través del transistor M1 siempre que las geometrías de los dos transistores sea la misma.

En esta configuración, el transistor M2 crea una copia de la corriente fluyendo a través del transistor M1 siempre que las geometrías de los dos transistores sea la misma. La derivación de la ecuación resultante es simple, debiendo sólo agregar la ecuación (3.9), en la ecuación del transistor, (3.8):

$$V_{d1} = V_{g1} = V_{g2}$$

$$V_{s1} = V_{s2}$$

$$I_2 = I_0 e^{\frac{V_T \ln\left(\frac{I_1}{I_0} + V_{s1} - V_{s2}\right)}{V_T}} = I_1 \qquad (3.10)$$

Una característica importante de los espejos de corriente es su habilidad de escalar las

corrientes que fluyen por ellos. La ecuación (3.10) asume que los dos transistores que forman el espejo de corriente tienen la misma geometría. Si el segundo transistor tiene un ancho o un largo más grande que el transistor operando como diodo, más o menos corriente fluirá a través de él con el mismo voltaje de gate. Recordando que el término I_0 en la ecuación del transistor fue utilizado para incluir la geometría de éste (W/L ec. (3.4)). Ahora si se elimina temporalmente este término de I_0 en la ecuación (3.8), se tiene que:

$$I = \frac{W}{L}\tilde{I}_0 e^{\frac{V_g - V_s}{V_T}} \tag{3.11}$$

Al escribir la ecuación (3.10) en estos nuevos términos, se obtiene:

$$I_2 = I_1 \frac{L_1 W_2}{L_2 W_1} \tag{3.12}$$

Aunque esta ecuación entrega una forma medianamente acertada de multiplicar una corriente por un valor fijo, presenta limitaciones, la principal es que se basa en hacer los transistores físicamente más pequeños o más grandes. Por ejemplo, para escalar una corriente al 10% de su valor, el ancho del transistor de salida debe ser el 10% del ancho del transistor en entrada, asumiendo que los largos se mantienen constantes. Esto indicaría que el ancho del transistor de entrada debe tener al menos 10 veces la mínima resolución del proceso.

3.3.1.4. Pares diferenciales

El par diferencial permite obtener una salida en corriente proporcional a la exponencial de su voltaje de entrada. Para esto, se asume que los transistores M1 y M2 están saturados y operando bajo-umbral. De la ley de corriente de Kirchhoff, es sabido que $I_1 + I_2 = I_b$. Luego, sustituyendo I_1 e I_2 en la ecuación (3.8), se tiene:

$$I_b = I_0 e^{-V}\left(e^{V_1} + e^{V_2}\right) \tag{3.13}$$

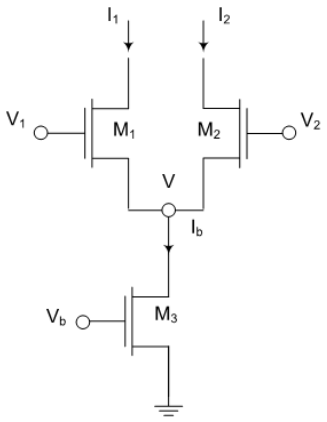


Fig. 3.5: Circuito esquemático de un par diferencial

La corriente de bias I_b es seteada por V_b y dividida entre I_1 e I_2 dependiendo de V_1 y V_2.

Resolviendo para V:

$$e^{-V} = \frac{I_b}{I_0\left(e^{V_1} + e^{V_2}\right)}$$

Sustituyendo la solución anterior en la ecuación (3.13):

$$I_1 = I_b \frac{e^{V_1}}{\left(e^{V_1} + e^{V_2}\right)} \tag{3.14}$$

$$I_2 = I_b \frac{e^{V_2}}{\left(e^{V_1} + e^{V_2}\right)} \tag{3.15}$$

Si luego se reemplaza I_b en las ecuaciones (3.14) y (3.15) con la ecuación (3.8), se obtiene:

$$I_1 = \frac{e^{V_b} e^{V_1}}{\left(e^{V_1} + e^{V_2} \right)} \tag{3.16}$$

$$I_2 = \frac{e^{V_b} e^{V_2}}{\left(e^{V_1} + e^{V_2} \right)} \tag{3.17}$$

Claramente se puede observar que el circuito presenta una respuesta similar a una del tipo exponencial, sin embargo posee los términos no deseados $e^{V_1} + e^{V_2}$ en el denominador de las ecuaciones (3.14) y (3.15).

3.3.1.5. Multiplicador de Gilbert

Existen una gran cantidad de metodologías para implementar multiplicadores en tecnología CMOS, una de ellas, está basada en el multiplicador de Gilbert de cuatro cuadrantes [8], el cual fue implementado originalmente con transistores bipolares. Los rangos lineales de este multiplicador cuando es operando bajo-umbral se limitan a aproximadamente 100 [mV], siendo bastante inferiores a los obtenidos operando sobre-umbral. El circuito esquemático del multiplicador de Gilbert utilizando transistores NMOS se muestra en la Fig. 3.6.

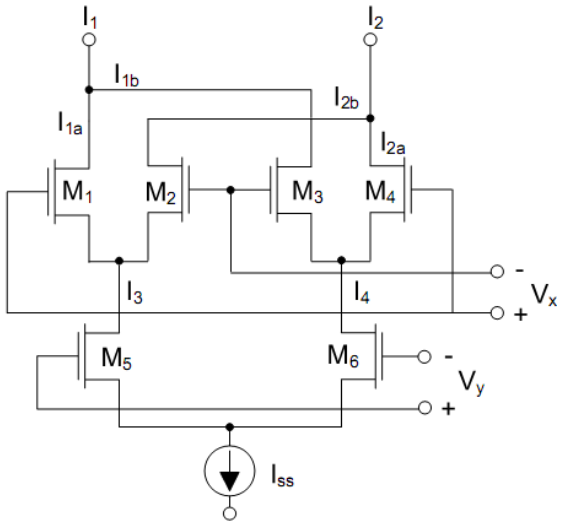


Fig. 3.6: Circuito esquemático del multiplicador de Gilbert.

La corriente de salida diferencial del multiplicador $\Delta I = I_1 - I_2$ es proporcional a la multiplicación de los voltajes diferenciales V_x y V_y.

Como se aprecia en la figura, el multiplicador está formado básicamente por tres pares diferenciales y una fuente de corriente I_{ss}, la que puede estar compuesta por un transistor o un espejo de corriente. El análisis de este circuito parte con el par diferencial compuesto por los transistores M5 y M6 y la fuente de corriente I_{ss}. Reescribiendo las ecuaciones (3.14) y (3.15) para agregar la corriente I_{ss} en lugar de I_b y considerando que $V_1 - V_2 = V_y$, se tiene:

$$I_3 - I_4 = I_{ss} \tanh\left(\frac{V_y}{2V_T}\right) \tag{3.18}$$

$$\cong I_{ss} \frac{V_y}{2V_T}, \text{ para } \left|I_{ss} \frac{V_y}{2V_T}\right| << 1$$

Luego, para los pares diferenciales restantes se tiene que:

$$I_{1a} - I_{2b} = I_3 \tanh\left(\frac{V_x}{2V_T}\right) \tag{3.19}$$

$$I_{1b} - I_{2a} = I_4 \tanh\left(\frac{-V_x}{2V_T}\right) \tag{3.20}$$

Por lo que la corriente de salida diferencial del multiplicador está dada por:

$$\begin{aligned}
\Delta I &= I_1 - I_2 \\
&= I_{1a} + I_{1b} - (I_{2a} + I_{2b}) \\
&= (I_{1a} - I_{2b}) - (I_{2a} - I_{1b}) \\
&= I_3 \tanh\left(\frac{V_x}{2V_T}\right) - {}^{-}I_4 \tanh\left(\frac{-V_x}{2V_T}\right) \\
&= \tanh\left(\frac{V_x}{2V_T}\right)(I_3 - I_4)
\end{aligned}$$

$$= I_{ss} \tanh\left(\frac{V_x}{2V_T}\right)\tanh\left(\frac{V_y}{2V_T}\right) \tag{3.21}$$

Como resultado del análisis, se obtiene que la característica de transferencia DC del multiplicador de Gilbert es el producto de las tangentes hiperbólicas de los dos voltajes de entrada. Luego, considerando que para $x < 0.5$,

$$\tanh(x) = x + \frac{x^3}{3} + \cdots \approx x,$$

si los voltajes de entrada V_x y V_y cumplen con la condición, $\frac{V_x}{2V_T} < \frac{1}{2}$ y $\frac{V_y}{2V_T} < \frac{1}{2}$ lo que arrastra a la condición $V_x < V_T$ y $V_y < V_T$, la ecuación puede ser simplificada como:

$$= I_{ss} \frac{V_x}{2V_T}\frac{V_y}{2V_T}$$

Por lo que la salida del multiplicador pasa a ser un valor proporcional a la multiplicación análoga de los voltajes de entrada.

$$\Delta I = I_{ss} \frac{V_x V_y}{4{V_T}^2} \tag{3.22}$$

En la Fig. 3.7, se muestra la curva de transferencia para un multiplicador de Gilbert implementado con transistores NMOS iguales, con razón W/L = 10, Vdd = 5 [V] e I_{ss} = 5 [μA]. El rango lineal del multiplicador se encuentra aproximadamente entre los -100 [mV] y 100 [mV]. Debido a que todos los transistores son iguales, no existe offset en el dispositivo.

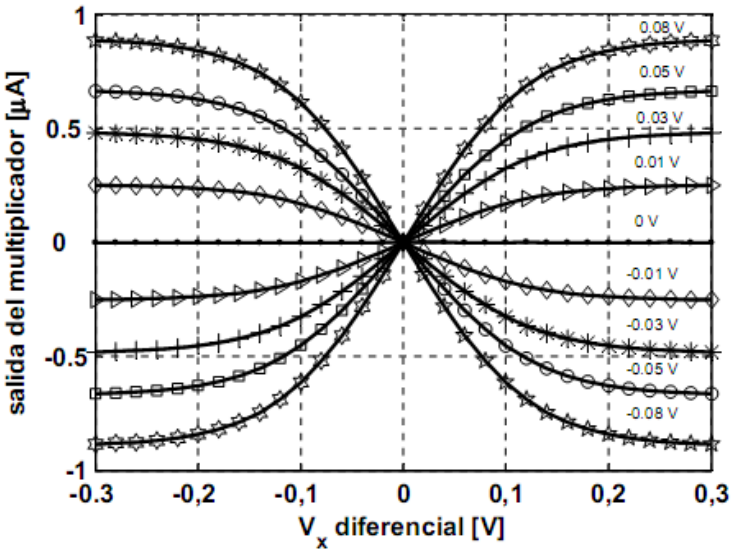


Fig. 3.7: Característica de transferencia DC de un multiplicador de Gilbert

Implementado con transistores NMOS operando bajo umbral.

3.4. Memorias para redes adaptivas

La celda de memoria es utilizada para almacenar los pesos sinápticos en el proceso de aprendizaje y en la implementación final del dispositivo. La utilización de una memoria digital en VLSI es costosa, tanto en área utilizada como en consumo de potencia, además sólo es posible almacenar información en forma de números binarios. Una alternativa es el uso de una memoria análoga para el proceso de aprendizaje.

Tanto en VLSI análogo como en digital existen dos tipos de almacenamiento disponibles: volátil y no volátil. El almacenamiento volátil es similar al de las RAM dinámicas, donde la carga es almacenada en un condensador conectado al gate de un MOSFET. Sin embargo, debido a las corrientes de fuga en las uniones p-n, la carga en los condensadores decae gradualmente, por lo cual el circuito *olvida* su valor almacenado. Debido a lo anterior, estos tipos de memoria requieren de frecuentes refrescos de sus valores para evitar la pérdida de datos. En la figura 4.9 se muestra el efecto de las corrientes de fuga en el almacenamiento de datos.

El efecto de las corrientes de fuga en las uniones p-n, ha sido el problema más difícil de solucionar para construir circuitos adaptivos eficientes. La adaptación y aprendizaje en tiempo real requiere de constantes de tiempo en el rango de [ms] a días. Debido a las corrientes de fuga en las uniones, la mayoría de los circuitos presentan constantes de tiempo menores a 1 [s], a menos que se

utilicen grandes capacitores con áreas prohibitivas.

Fig. 3.8: Almacenamiento dinámico de valores análogos en un condensador

(a) Circuito esquemático ilustrando el almacenamiento análogo en un condensador y la corriente de fuga a través de la unión p-n. **(b)** Gráfico del voltaje almacenado v/s tiempo. El voltaje decrece linealmente en el tiempo debido a que la corriente de fuga es constante.

El almacenamiento no volátil utiliza tecnología de compuerta flotante (floating-gate) para almacenar datos indefinidamente sin necesidad de refrescarlos. Esta tecnología es utilizada en Memorias EEPROM (Electrically Erasable Programmable Read Only Memory). Este tipo de memorias no sólo tienen la ventaja de almacenamiento a largo plazo, sino que además presenta una mayor eficiencia de recursos que las memorias volátiles, debido a que no requieren de circuitería adicional para refrescar los valores almacenados.

3.4.1. Celda de memoria basada en transistores sinápticos

Un dispositivo que hace uso de la tecnología de compuerta flotante es el denominado transistor sináptico [1], [5]. Este dispositivo provee una forma compacta, no volátil y análoga de almacenar carga en su compuerta flotante y proporciona mecanismos locales para actualizarla en forma precisa durante la operación normal. Además, utiliza un área menor que memorias análogas volátiles y registros digitales. Debido a estas características, los transistores sinápticos han sido una elección popular para el almacenamiento de pesos en sistemas de aprendizaje en silicio desarrollados en el último tiempo.

La Fig. 3.9 muestra el layout de un transistor sináptico en un proceso de doble poly y el símbolo utilizado para representarlo en circuitos esquemáticos [1]. El dispositivo está compuesto por dos transistores pFET (M1 y M2), los que comparten su compuerta flotante y almacena un valor no

volátil representado por la carga en el gate poly1.

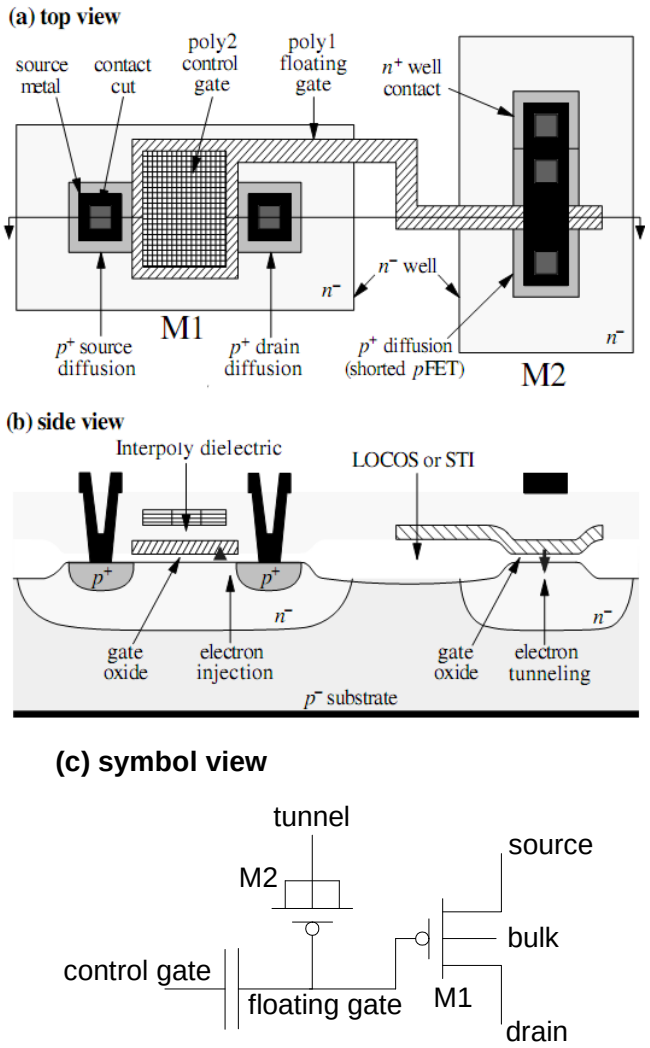


Fig. 3.9: Transistor sináptico

En (**a**) y (**b**) se muestra el layout del circuito. El dispositivo permite almacenamiento no volátil de carga en la compuerta flotante del transistor pFET M1. El pFET M2 actúa como una juntura de túnel (tunneling junction), y es utilizado para remover electrones desde la compuerta flotante. Para agregar electrones a la compuerta, se utiliza la técnica impact ionized hot-electron injection (IHEI) desde el drain de M1.

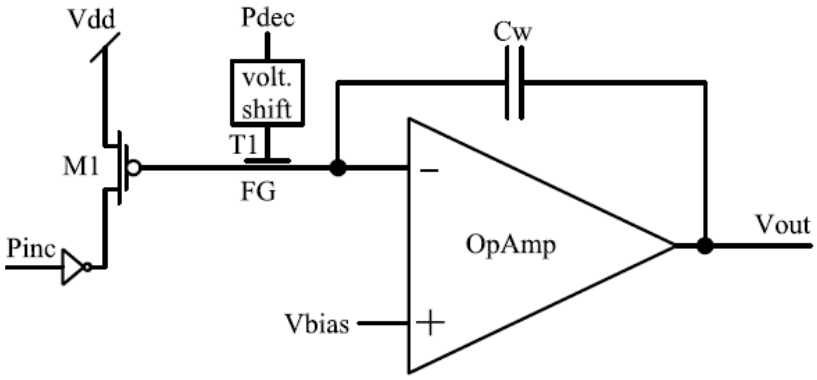
Los métodos utilizados para modificar la carga almacenada en la compuerta flotante del transistor sináptico están regidos por dos procesos, *Fowler-Nordheim Tunneling* e *Impact ionized Hot-Electron Injection*. Ambos mecanismos permiten una actualización precisa del valor almacenado.

El proceso *Fowler-Nordheim Tunneling* es utilizado para remover electrones de la compuerta flotante; una diferencia de potencial entre la unión de tunneling y la compuerta flotante causa que los electrones salgan de la compuerta a través del óxido del gate del pFET al n-well, siendo la magnitud de esta corriente dependiente del voltaje en el óxido.

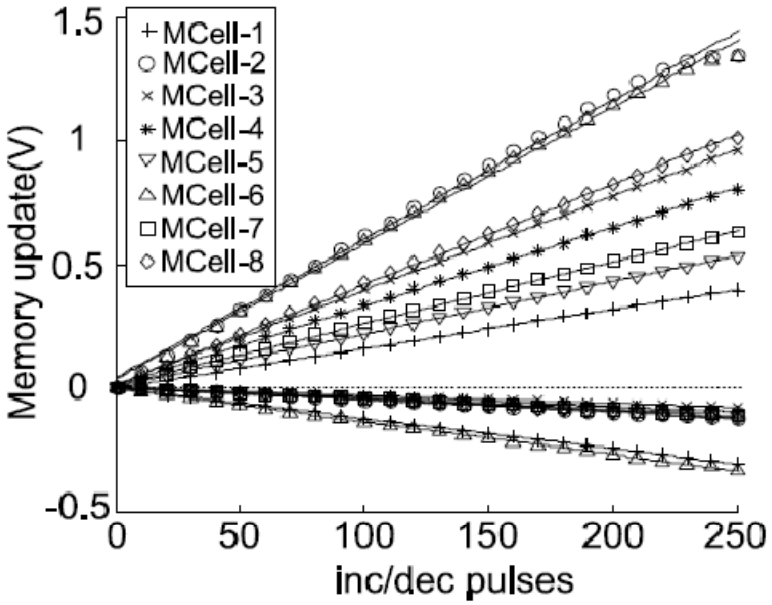
El proceso IHEI [4] permite agregar electrones a la compuerta flotante. La forma en que esto se realiza es la siguiente: los huecos del canal se ven acelerados en la región de agotamiento y colisionan con las paredes del semiconductor. Cuando el campo eléctrico canal-drain es grande, una fracción de estos huecos choca con suficiente energía como para liberar pares electrón-hueco adicionales. Los electrones ionizados, promovidos a su banda de conducción por esta energía, son expulsados del drain por el mismo campo eléctrico. De esta forma los electrones expulsados con una energía cinética superior a 3.1 eV pueden ser inyectados en la banda de conducción del óxido y ser aceptados por la compuerta flotante.

A pesar de las ventajas mencionadas anteriormente, se hace difícil implementar reglas de aprendizaje lineales. Esto se debe a que las dinámicas que poseen estos procesos son de naturaleza exponencial con respecto a sus variables de control, las cuales llevan a una dependencia no-lineal de las actualizaciones. Esto es un problema sumamente importante debido a que el desempeño del algoritmo de aprendizaje está fuertemente relacionado con la exactitud en la actualización en los pesos, por lo tanto distorsiones en la regla de aprendizaje degradan fuertemente su desempeño.

Una forma de solucionar este problema es utilizar la celda de memoria de la Fig. 3.10 (a) descrita en [2]. Su funcionamiento se basa en almacenar el valor análogo, correspondiente al peso sináptico, como carga en la compuerta flotante FG del transistor sináptico M1. Pulsos de amplitud y ancho fijo en Pdec y Pinc activan el tunneling y la inyección agregando o removiendo carga desde la compuerta flotante, respectivamente.



(a) Celda de memoria con actualizaciones lineales



(b) Actualizaciones en ocho celdas de memoria

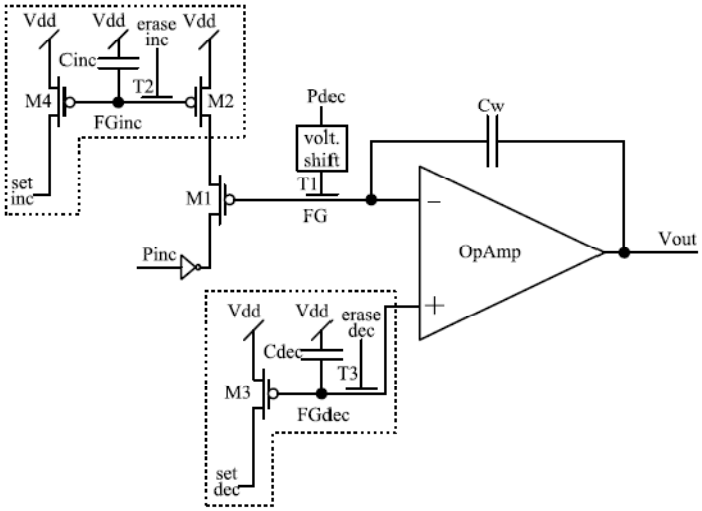
Fig. 3.10: Celda de memoria análoga modulada por densidad de pulsos (PDM)

(a) Cada peso es almacenado como carga análoga no volátil en la compuerta flotante FG. Las actualizaciones del peso son linealmente proporcionales a la densidad de los pulsos de P_{inc} y P_{dec}. **(b)** Actualizaciones de memoria en ocho celdas como función de la densidad de los pulsos de actualización. Las líneas con pendiente positiva corresponden a pulsos P_{inc}, mientras que las actualizaciones negativas corresponden a pulsos P_{dec}. Las actualizaciones poseen una linealidad superior a 10-bit, pero presentan diferencias considerables entre las actualizaciones positivas y negativas de una misma celda como entre celdas diferentes, lo cual se traduce en actualizaciones asimétricas y no-uniformes.

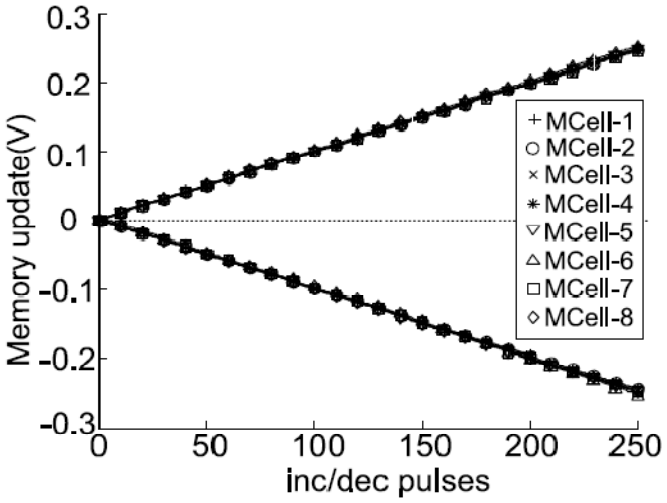
El amplificador operacional fija el voltaje de la compuerta flotante al voltaje global V_{bias}. El condensador C_w integra las actualizaciones de la carga, cambiando la salida V_{out} por $\Delta V_{out} = \Delta Q / C$.

Debido a que el voltaje en la compuerta flotante es constante, al igual que la amplitud y ancho de los pulsos, la magnitud de las actualizaciones depende de la densidad de los pulsos P_{inc} y P_{dec}. La Fig. 3.10 (b) muestra la función de transferencia de ocho celdas de memoria implementadas en un solo chip. Aquí es posible apreciar otro importante problema: el mismatch en el dispositivo hace imposible lograr actualizaciones simétricas en una celda o actualizaciones iguales a través de diferentes celdas de memoria. Sin actualizaciones simétricas, el error residual se incrementa significativamente, mientras que actualizaciones diferentes a través de las celdas llevan a una convergencia más lenta de los algoritmos adaptivos.

En la Fig. 3.11 (a) se muestra un diseño mejorado [2] de la celda de memoria anterior, en la que se incorporan mecanismos de calibración local para solucionar los problemas de asimetría y no-uniformidad en la actualización de los pesos. El voltaje en la compuerta flotante FG_{dec} fija el voltaje en FG y controla la razón entre la fuerza de tunneling e inyección sobre FG: incrementando el voltaje en FG se debilita el tunneling y se fortalece la inyección y viceversa.



(a) Celda de memoria con calibración en chip



(b) Actualizaciones simétricas y uniformes en ocho celdas de memoria

Fig. 3.11: Celda de memoria análoga con calibración en el chip

(a) El voltaje en la compuerta flotante FG_{inc} controla la fuerza de la inyección de electrones sobre FG. El voltaje en FG_{dec} controla la razón entre la fuerza de tunneling e inyección, y puede ser usado separadamente para lograr actualizaciones simétricas en una celda, o en conjunto con FG_{inc} para obtener actualizaciones simétricas y uniformes a través de todo el chip. **(b)** Actualizaciones de las celdas de memoria después de la calibración global realizada sintonizando los voltajes en FG_{inc} y FG_{dec}.

Este voltaje puede ser usado para lograr actualizaciones simétricas en una celda, o en conjunto con FG_{inc} para obtener actualizaciones simétricas y uniformes a través de todo el chip. La figura Fig. 3.11 (b) muestra las actualizaciones para las celdas de memoria después de la calibración global realizada sintonizando los voltajes FG_{inc} y FG_{dec}.

3.5. Simulador de Circuitos Adaptivos

La creación del emulador para sistemas adaptivos se basa en los principios del simulador existente, es por esto que se revisan los aspectos generales del funcionamiento de este, para luego mostrar un ejemplo de aplicación.

La herramienta de simulación está compuesta por 7 funciones desarrolladas en Matlab®. Algunas de estas funciones requieren del uso del software LTspice para generación de características de transferencia mediante simulación de circuitos.

3.5.1. Fundamentos de operación

La herramienta de programación se basa en el uso de las curvas de transferencia *dc* de los circuitos utilizados como unidades aritméticas y de procesamiento en VLSI. Estas curvas de transferencia representan un mapeo estático entrada-salida de los dispositivos, y son utilizados para reemplazar las operaciones aritméticas y de procesamiento ideales.

El proceso para incorporar los efectos del hardware en las simulaciones consta de varias etapas. La primera de ellas busca obtener un modelo de entrada-salida de un dispositivo utilizado en VLSI, para ser utilizado en reemplazo de las funciones ideales. Para esto es necesario el estudio de circuitos que permitan implementar una función aritmética o de procesamiento en particular. Luego de probar la respuesta del circuito, a través de herramientas CAD de simulación, se exporta la característica para construir las expresiones de entrada y salida de la característica.

Los pasos para llevar esto a cabo se enumeran a continuación.

1. Estudiar los circuitos que permitan realizar los cálculos aritméticos o de procesamiento requeridos para implementar algún esquema adaptivo.
2. Simular mediante software CAD, para la obtención de la característica *dc*.
3. Exportar y seleccionar las variables útiles para la creación de la característica de transferencia.
4. Creación de la característica de transferencia del circuito.

Otra forma de obtener la característica de transferencia de un dispositivo es a través de *la medición directa de las variables del circuito*. De esta forma, la herramienta de simulación tiene la capacidad de obtener la característica de transferencia *dc* de un dispositivo, a partir de datos de laboratorio.

Al finalizar las etapas ya descritas, se obtiene una estructura de datos que contiene la curva de transferencia *dc* del dispositivo modelado.

La siguiente etapa busca incorporar la característica de transferencia del dispositivo en el algoritmo mismo. Para esto, son necesarias dos cosas. La primera es escalar los datos de la característica a un rango numérico, debido a que están dados en unidades eléctricas. La segunda es

crear las funciones que permitan incluir estas características de transferencia en los cálculos aritméticos y de procesamiento.

Una característica adicional muy útil es poder incluir los efectos del *device mismatch* en los componentes del circuito. Para realizar esto, es necesario agregar variaciones aleatorias de offset y ganancia en las características de los dispositivos. Esto permite además crear características de transferencia diferentes a partir del mismo circuito.

Los pasos que se deben seguir en esta etapa se enumeran a continuación:

1. Agregar variaciones aleatorias a las curvas de transferencia, si es necesario.
2. Escalar los datos de la característica de transferencia a un rango numérico adecuado para el funcionamiento del algoritmo.
3. Desarrollar funciones que permitan reemplazar el cálculo aritmético y procesamiento ideal, por uno que se realice a través de la característica de transferencia de los circuitos.

A continuación se demuestra una aplicación que ejemplifica las funciones utilizadas en el simulador de software.

3.5.2. Ejemplo

Se desea simular el comportamiento de una red neuronal de 11 sinapsis utilizando un esquema como en la Fig. 2.1. El modelo de aprendizaje de pesos a utilizar es LMS, los pasos a seguir para simular el rendimiento de esta red se detalla en los siguientes pasos:

1. Se construye las características de transferencia de 11 multiplicadores, para esto se utiliza la herramienta de simulación que toma los datos de mediciones de laboratorio.
2. Con la estructura representando las 11 características de transferencia de los multiplicadores escogidos; se utiliza la herramienta de simulación que escala los valores de estos multiplicadores a rangos numéricos definidos por el usuario, con el fin de trabajar en un rango diferente y da la posibilidad de trabajar en un ambiente familiar con los valores comunes aplicables a redes neuronales.
3. Se obtienen, gracias a herramienta de simulador, las características de transferencia de 11 memorias, manteniendo la relación de una memoria por un multiplicador en la simulación de

las sinapsis. Se verifica además que el rango operativo de valores sea el mismo utilizado en el multiplicador, para no generar discordancias.

4. Se realiza la iteración, empezando por el cálculo del forward path, en donde se obtiene la salida, gracias a la suma del resultado de cada multiplicador, estos resultados se consiguen con la función del simulador que utiliza la característica de transferencia de cada multiplicador, para la entrega de productos respetando los inconvenientes de implementación.
5. Se procede con el cálculo de feedback path, para esto se utiliza función de simulador encargado de tomar las propiedades de cada memoria y realizar la actualización apropiada, tomando en cuenta el error, la tasa de aprendizaje y el vector de entrada.

Bajo estos procedimientos se construye y se simula una red adaptiva, luego de finiquitar las iteraciones del forward path y feedback path, se realizan las observaciones que determinan si el riesgo de implementar un algoritmo con las características esperadas es menor o requiere ciertas técnicas de compensación previas.

Capítulo 4. Emulador de circuitos adaptivos

4.1. Introducción

Bajo los pilares en donde descansan los principios expuestos para algoritmos de adaptación y hardware disponible para su implementación en VLSI, se dispone a exhibir los aspectos de diseño del emulador de circuitos adaptivos análogos, en la sección 4.2 se presenta la arquitectura que representa de forma general los conceptos encerrados en la emulación, en la sección 4.3 se revisa el diseño para lograr la funcionalidad deseada, posteriormente en la sección 4.4 se implementa en lenguaje de descripción de hardware Verilog HDL, cuidando que esta sea compatible con los requerimientos operativos de la tarjeta Virtex II Pro.

4.2. Arquitectura Emulador

El funcionamiento en términos generales de este emulador es evidenciado en la arquitectura de lo que se desea lograr como meta, la propuesta funcional corresponde a un filtro adaptivo utilizando el algoritmo LMS para el entrenamiento de los pesos. La estructura se muestra en la Fig. 4.1.

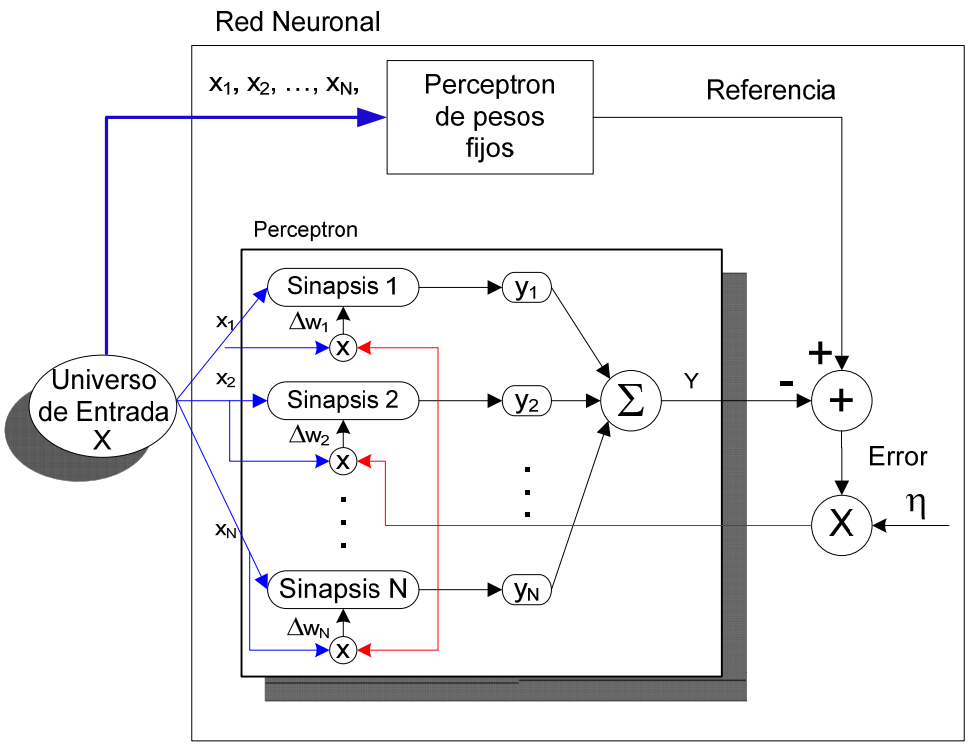


Fig. 4.1: Arquitectura Emulador

Para efectos de simplificación con respecto al esquema de Fig. 2.2, el emulador no reproduce ruido Gaussiano, se asume que éste se aplica explícitamente a la entrada de la referencia. La estructura recibe desde el universo de entrada valores que son ponderados en los bloques que representan las sinapsis, compuestas por un multiplicador y una celda de memoria. Estas celdas de memoria y multiplicadores reflejan el funcionamiento de las características de transferencia de las muestras de laboratorio obtenidas por el profesor Dr. Miguel Figueroa. Las salidas de cada sinapsis corresponden a cada término del producto interior $Y = x^T w$. Esta salida es comparada con la referencia (la cual es constituida en el emulador para comprobar su funcionamiento) para generar un error, el cuál es utilizado para el entrenamiento de los pesos.

En conjunto con la aplicación de la tasa de aprendizaje, se aplican los cambios a cada celda de memoria, que terminan cambiando los valores de los pesos contenidos en cada sinapsis, reforzando y debilitando según sea la adaptación a seguir.

En el esquema de la Fig. 4.1, se demuestra la función del perceptrón y la red que entrena los enlaces sinápticos, pero no se aprecia una función detallada de las sinapsis presentes, en la Fig. 4.2

se puede observar con mayor profundidad la labor que esta desempeña.

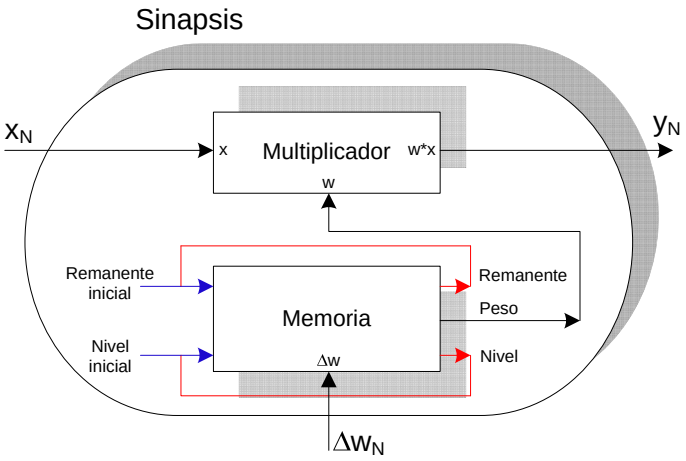


Fig. 4.2: Arquitectura Sinapsis

Como bien se explica en el párrafo anterior, cada sinapsis está compuesta por un multiplicador y memoria configurados como se muestra en la Fig. 4.2. Haciendo una evaluación más exhaustiva, la memoria contiene realimentaciones para el cálculo del remanente y nivel de actualización, esto en base al delta de actualización aplicado, el peso derivado de esta operación, va conectado directamente al multiplicador correspondiente. Este inicia la multiplicación y obtiene el resultado para ser sumado con las salidas de las sinapsis restantes, se destaca de esta configuración la similitud con las celdas de memoria/multiplicador reales.

Continuando con la introducción de tipo funcional de las estructuras, se procede a presentar el multiplicador que compone cada sinapsis en este emulador, el cual se en la Fig. 4.3.

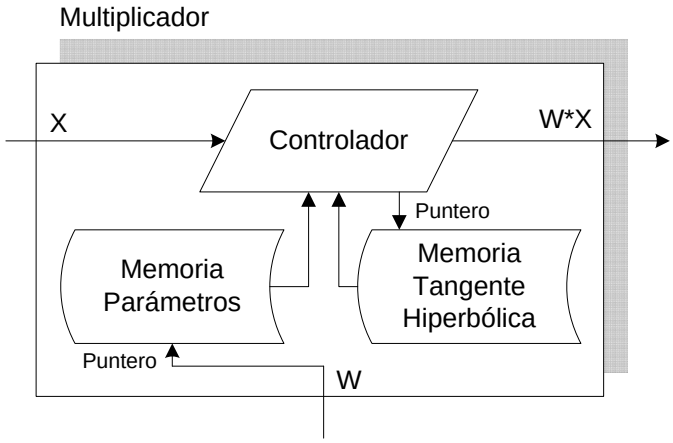


Fig. 4.3: Arquitectura Multiplicador

Como bien se puede notar, la constitución del multiplicador incluye dos módulos de memoria para su correcto funcionamiento. Estas constan por un lado de un módulo de memoria de parámetros que almacena las constantes representativas de las curvas de característica. Esto se detallará en profundidad en la sección 4.3, el otro módulo de memoria almacena parámetros para la obtención de la forma de la tangente hiperbólica, ya que la función tanh no existe y la implementación en FPGA requiere la utilización de recursos que pueden ser utilizados más eficientemente en lógica. Este punto también será discutido con mayor profundidad en la siguiente sección.

El controlador que aparece en la Fig. 4.3 se encarga de tomar los datos desde los módulos de memoria, de los cuales uno es apuntado directamente por el peso que ingresa como multiplicador (módulo memoria parámetros), y el otro por el controlador mismo. La base del funcionamiento del controlador es tomar ambos multiplicandos (x y w), consultar por los parámetros correctos para emular un multiplicador análogo y entregar como resultado lo mismo que entregaría con multiplicador análogo integrado en VLSI.

Revisando nuevamente la sinapsis de la Fig. 4.2, se observa que la memoria incluida provee diferentes funciones a las observadas inicialmente en la integración del multiplicador, esta memoria se encarga de gestionar el peso que es utilizado en la ejecución de una red adaptiva, en la Fig. 4.4 se muestra el esquema de funcionamiento.

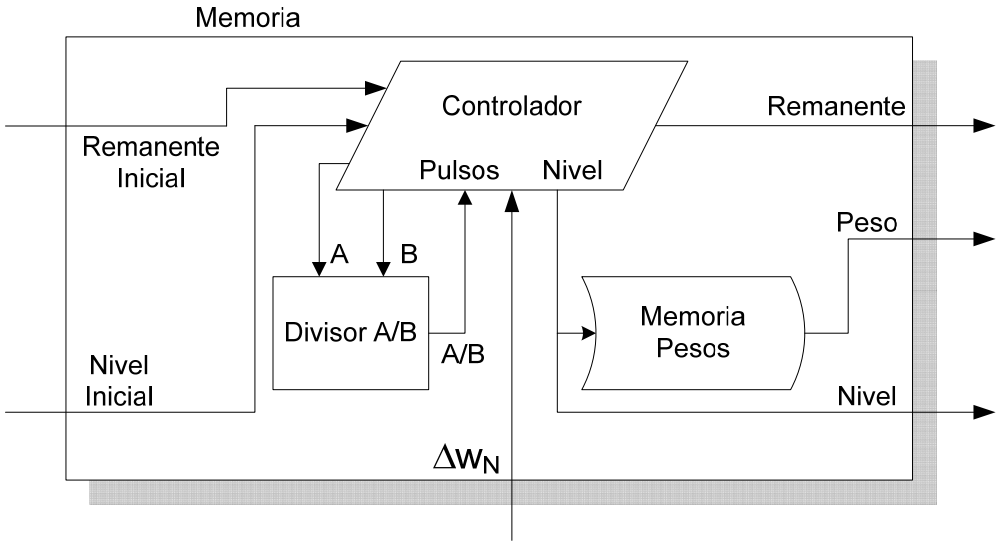


Fig. 4.4: Arquitectura Memoria

El controlador de este módulo tiene la función de tomar los datos iniciales de la memoria a emular, para esto primero recibe el nivel inicial, este es indicado como un índice el cuál debe tener rangos admisibles para la memoria a utilizar, representado por el nivel, además requiere de la actualización que se le quiera dar a esta celda, representada por ΔW_N, finalmente la operación necesita conocer el remanente que pudo haber quedado de alguna actualización anterior, el cual está representado por el remanente inicial.

Luego de reconocer las entradas, el controlador es capaz de determinar la cantidad de pulsos necesarias a aplicar utilizando el modulo de división, el cuál será profundizado en la sección siguiente, con la cantidad de pulsos se puede calcular el nuevo nivel al cual será actualizada la celda de memoria; además se tendrá el remanente, y utilizando el módulo de memoria con la característica de la celda, se accederá al nuevo valor de peso. Hay que destacar que el remanente y nivel de salida son realimentadas como se muestra en la Fig. 4.2. Esto con el fin de afirmar la condición de actualización para futuras adaptaciones de pesos.

Con las bases vistas de la arquitectura a desarrollar en el emulador, se procede a especificar el diseño de cada uno de estos módulos.

4.3. Diseño Emulador

El diseño del emulador tiene como pilar fundamental la transferencia de información y operaciones siguiendo la estructura de una máquina de estados, las razones de esta alternativa de diseño es por facilidad de entendimiento de lo que se desea obtener, y ordenamiento de objetivos. A continuación se detallarán los aspectos revisados en la síntesis del emulador, siguiendo el principio antes enunciado.

4.3.1. Red Neuronal

Observando la arquitectura de la Fig. 4.1, se puede deducir que el bloque principal se compone principalmente por dos perceptrones, uno de pesos fijos y otro de pesos adaptivos, este bloque presenta a ambos perceptrones un vector del universo de entrada, obtiene el error correspondiente a las salidas y aplica la actualización de pesos al perceptrón adaptivo, los pasos que debe aplicar son:

1. Activar los multiplicadores de ambos perceptrones
2. Luego de tener los resultados de cada perceptrón fijo, calcular referencia, correspondiente a la suma de las salidas del perceptrón de pesos fijos.
3. Verifica que el perceptrón de pesos adaptivos tiene su salida y calcula el error.
4. En base al error y la tasa de aprendizaje, se envían las actualizaciones de pesos al perceptrón para que este se encargue de actualizar las sinapsis adaptivas.

El esquema de estados que muestra estos principios se refleja en la Fig. 4.5.

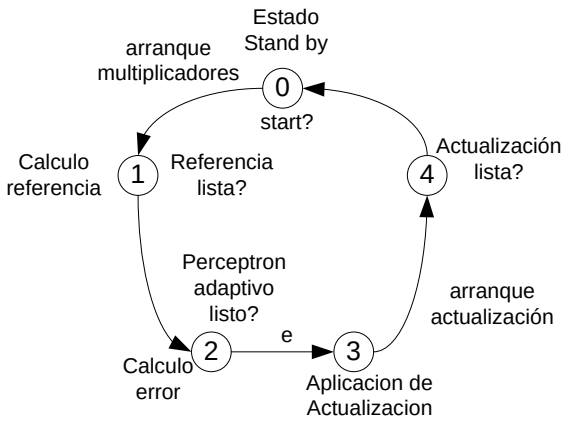


Fig. 4.5: Máquina de estados red neuronal

El estado inicial (0) corresponde al estado de espera, en donde la partida es otorgada por un dispositivo externo, luego del arranque se inician las multiplicaciones en ambos perceptrones, el estado 1 espera que el perceptrón de pesos fijos obtenga sus resultados, con eso se obtiene la referencia gracias a la suma de los resultados de cada multiplicador, posteriormente se produce la transición al estado 2, donde se verifica si el perceptrón adaptivo contiene la salida, cuando esto ocurre se calcula el error, lo que produce el cálculo de la actualización de pesos en el estado 3, finalmente en el estado 4 se verifica que la actualización se lleva a cabo.

4.3.2. Perceptrón

El perceptrón adaptivo que está representado en la sección anterior obedece su operación a las interacciones con la red neuronal, en donde recibe la activación para multiplicar el peso almacenado en cada sinapsis con el vector del universo de entrada, además obtiene la salida para ser comparada con la referencia, luego recibe desde la red neuronal la activación para actualización de pesos, concluye el funcionamiento del perceptrón indicando cuando la actualización se lleva a cabo en todas las memorias. La peculiaridad de este módulo es la necesidad de diseñar dos pequeñas máquinas de estados, para operar los multiplicadores y las memorias, los cuales se muestran en la Fig. 4.6.

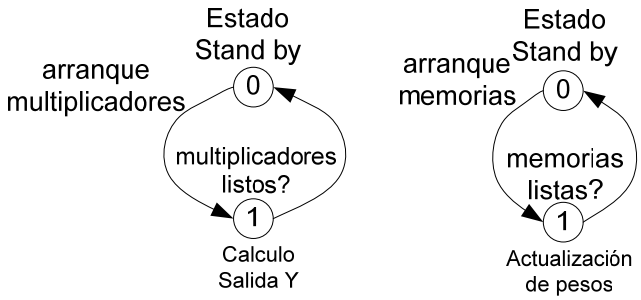


Fig. 4.6: Máquinas de estado perceptrón

Los estados requeridos para cada operación son sólo dos, ya que el estado de operación (1) es suficiente para la espera que todas las memorias o multiplicadores entreguen su resultado, esto se realiza por medio de la sinapsis la cual es mostrada a continuación.

4.3.3. Sinapsis

La función de la sinapsis es agrupar la celda de memoria con el multiplicador correspondiente, esto con el fin de no mezclar memorias con multiplicadores de diferentes características. Además en este módulo se realizan las realimentaciones de la memoria para actualizar la celda y pueda ser esta usada en la siguiente iteración. Este apartado no tiene una base de diseño, solo obedece la arquitectura mostrada en la Fig. 4.2.

4.3.4. Multiplicador

La emulación del multiplicador corresponde a una de las bases del trabajo, en ella descansan una gran variedad de instrucciones necesarias para poder realizar de forma satisfactoria la operación de reproducir el comportamiento de un multiplicador análogo. Antes de introducir el diagrama de diseño, se profundiza el principio de síntesis.

El multiplicador emulado para desempeñarse como su par VLSI análogo, requiere barrer de forma similar las curvas características. Para llevar esto a cabo, se requieren transferir las características de transferencia desde mediciones de laboratorio almacenadas en Matlab®, correspondientes en este caso a 64 multiplicadores. Cada multiplicador análogo realiza distintos barridos en su multiplicación, esto se debe a que uno de los operandos proviene de una memoria análoga, y la capacidad de la memoria para incrementar o decrementar su peso varían fuertemente de una memoria a otra. La Fig. 4.7 muestra el esquema de celdas de multiplicación y memoria

análoga integradas, el barrido se realiza primero incrementando o decrementando el peso almacenado en la memoria (mediante P_{INC} y P_{DEC}), luego se realiza un barrido en x, el cual tiene una cantidad de muestras fijas para todos los multiplicadores.

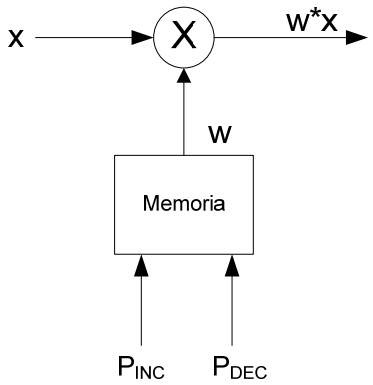


Fig. 4.7: Celda multiplicador y memoria análoga

Con los datos provistos de laboratorio, el barrido en x consta de 41 puntos y los barridos en w varían en cada multiplicador, con valores de 37 hasta 367 puntos.

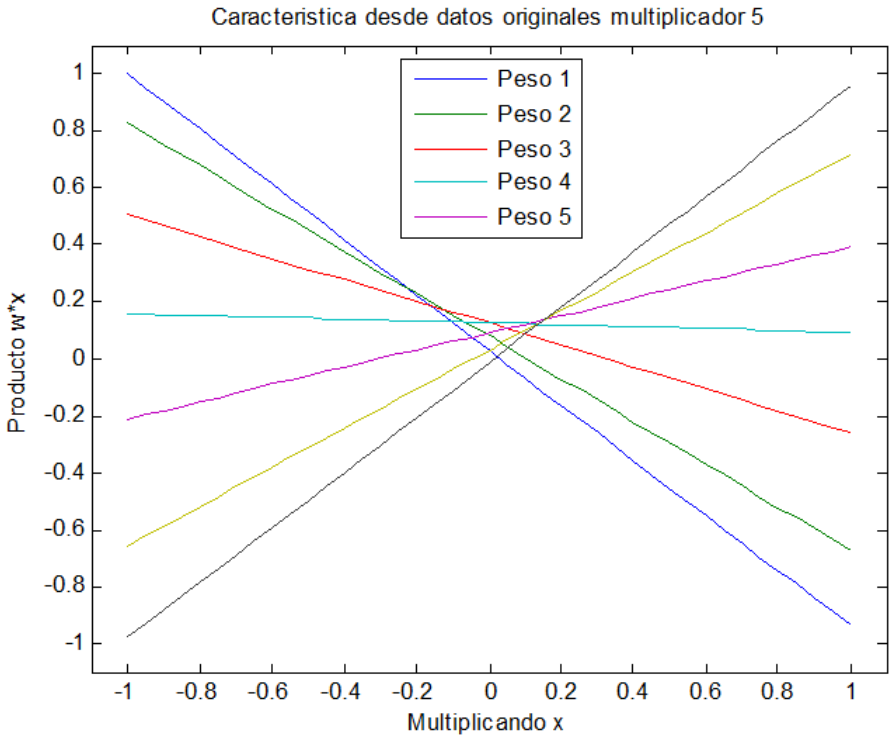


Fig. 4.8: Característica original de transferencia, multiplicador 5

Si se quisieran almacenar todos los puntos en los barridos para todos los multiplicadores, se requerirían grandes cantidades de bloques de memoria ROM. Para evitar el consumo masivo de recursos, se decide aproximar cada barrido en w, que se asemeja a una tangente hiperbólica (ejemplo en Fig. 3.7), por una representación del tipo,

$$w \cdot x = a \cdot \tanh\left(b \cdot x\right) + c \tag{4.1}$$

donde a, b y c son constantes que aproximan las curvas para cada peso, si se revisa la ec. (3.21), se puede encontrar la similitud con los parámetros *a* y *b*, los cuales son:

$$a = I_{ss} \tanh\left(\frac{V_y}{2V_T}\right)$$

$$b = \frac{1}{2V_T}$$

Todo esto asumiendo que el voltaje *x* es aplicado a V_x y el peso *w* es aplicado a V_y. La constante *c* por otro lado, representa el offset presente en cada multiplicador, debidos al *mismatch* comentado anteriormente, que afecta a unos pesos más que en otros, los cuales se ven evidenciados en los datos de laboratorio para el multiplicador 5 en la Fig. 4.8.

Luego de extraer los parámetros *a*, *b* y *c* para aproximar los datos de laboratorio, se observa que para utilizar estas constantes en una implementación en FPGA, es requerimiento contar con la función tangente hiperbólica. Hasta este punto, no se cuenta con esta característica implícita que pueda ser utilizada en un diseño. Por lo tanto este aspecto forma parte también del diseño para la emulación del multiplicador.

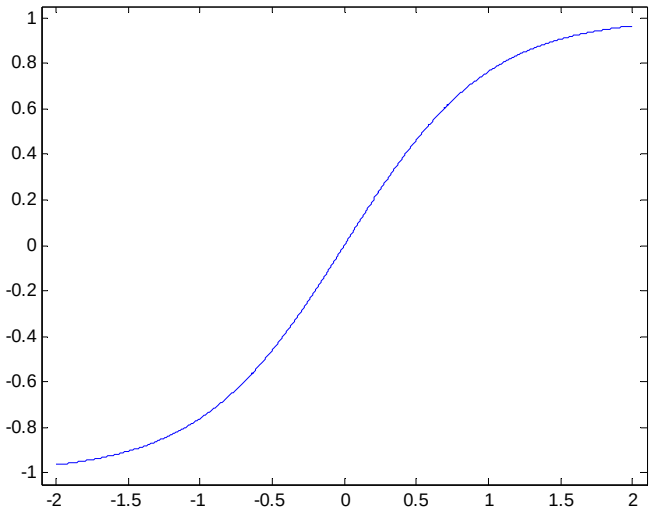


Fig. 4.9: Curva tangente hiperbólica

En la Fig. 4.9, se muestra la función tangente hiperbólica graficada en ambiente Matlab®, las formas de poder trabajar con esta función en FPGA son variadas, algunas opciones de diseño pueden ser como se indican a continuación:

1. Construir la función mediante aproximaciones de Taylor.
2. Almacenar cierta cantidad de puntos de la función en memoria.
3. Aproximar mediante rectas pequeños tramos.

Estos tres acercamientos tienen sus ventajas y desventajas, por ejemplo la primera opción puede entregar valores muy precisos, más aún si se trabaja cerca del origen x=0, pero la implementación puede resultar compleja por lo cual consumiría muchos recursos al tener términos de orden superior. La segunda alternativa por otro lado, no requeriría operación adicional al módulo de memoria, pero la necesidad de precisión podría elevar a valores no permisibles la memoria requerida para lograr la premisa. Por último se revisa esta tercera alternativa que representa un paso intermedio entre las opciones anteriores, esto porque la aproximación por rectas requiere el almacenamiento de parámetros de rectas, pero su precisión no se ve afectada de gran manera al calcular un punto ya que realiza indirectamente una interpolación y no se requiere incrementar la memoria en grandes cantidades para tener mejorías apreciables de precisión, es por esto que se

recurre a esta 3ra opción para complementar los parámetros *a*, *b* y *c* en el diseño del multiplicador.

El esquema de esta alternativa radica en el almacenamiento de los parámetros necesarios para la generación de rectas, si se revisa la ecuación de la recta,

$$y = m \cdot \left(x - x_o \right) + y_o \tag{4.2}$$

Se puede deducir los parámetros necesarios, los cuales son,

$$m = \frac{y - y_0}{x - x_0} \tag{4.3}$$

$$y_0 - m \cdot x_0 = m \cdot x - y \tag{4.4}$$

Con lo cual se puede utilizar posteriormente la ecuación de la recta de una forma más práctica:

$$y = m \cdot x + b_y \tag{4.5}$$

donde,

$b_y = y_0 - m \cdot x_0$, representa el offset de *y* cuando *x* es cero.

El esquema de diseño se muestra en la Fig. 4.10, en el diagrama se representan los estados necesarios para realizar las distintas operaciones para la obtención del resultado equivalente al multiplicador real.

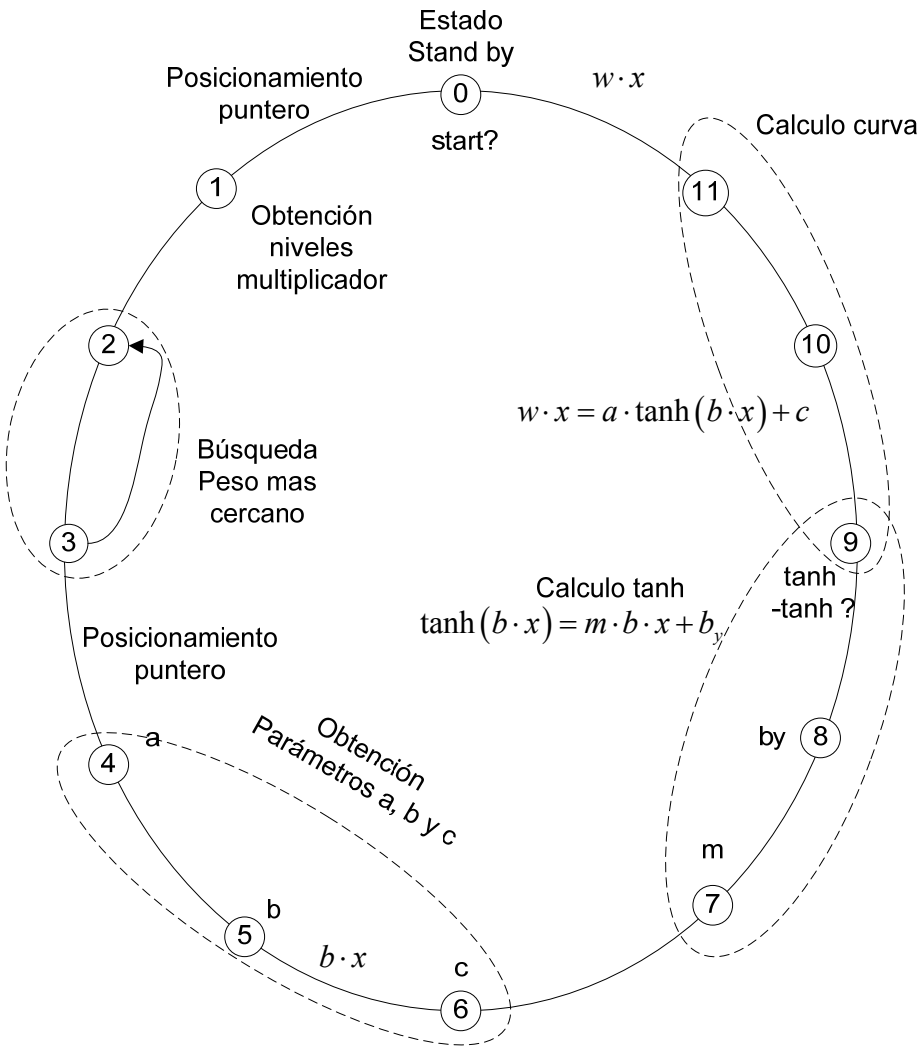


Fig. 4.10: Máquina de estados multiplicador

Los pasos que debe realizar el multiplicador se enuncian a continuación:

1. Espera de arranque de multiplicación por parte de la sinapsis a la cual está conectada, inicializa al mismo tiempo los valores que se utilizarán en futuros estados.
2. Se obtiene cantidad de niveles de peso w, esto con el fin de iniciar la búsqueda del peso más cercano al ingresado como multiplicando.
3. Se realiza búsqueda de peso más cercano, luego que se encuentra el peso que se asemeja más al ingresado a multiplicar, se pueden obtener los parámetros *a*, *b* y *c* que describen la forma de la curva a utilizar.

4. Al tiempo que se obtiene el parámetro *b*, se calcula inmediatamente $b \cdot x$ y se extraen los parámetros *m* y b_y para la interpolación usando la ecuación de la recta.
5. Se obtiene la tangente hiperbólica y se condiciona según signo de x (ver sección implementación).
6. Se aplican constantes *a* y *c* para cálculo final de producto $w \cdot x$.

Con esta plataforma definida se puede iniciar la implementación del módulo, a continuación se muestra los aspectos de diseño de la memoria a utilizar en conjunto con el multiplicador.

4.3.5. Memoria

Recogiendo varios conceptos vistos en el diseño de la emulación del multiplicador análogo, se observa la descripción del funcionamiento de una memoria análoga. En punto anterior y en 3.4.1, se menciona que el funcionamiento de una celda de memoria es a bases de pulsos de incremento y decremento (P_{INC} y P_{DEC}).

Para aplicar entonces una actualización de pesos a una celda, es necesario conocer cuántos pulsos se requieren aplicar para llevar un valor de peso a otro deseado, por lo que debe hacer de primera mano, es retener el menor incremento entre un peso y su valor próximo al aplicar un pulso (Δw mínimo); luego de obtener este incremento/decremento, se requiere determinar la cantidad de pulsos que generaran el Δw en la celda de memoria; para esto se requiere hacer una división, función que no es fácilmente construible para síntesis en FPGA; gracias a la teoría expuesta en 0, se diseña la función capaz de obtener la cantidad de pulsos, la síntesis de esta se detalla al final de esta sección.

Los pesos de la memoria, que se encuentran almacenados en las mediciones de laboratorio de un archivo en ambiente Matlab®, son traspasados a Verilog HDL para acceder a estos en tiempo de ejecución; los datos almacenados en este modulo de memoria corresponde a la cantidad de niveles que se pueden acceder, los valores de los pesos como tal y el mínimo incremento que genera un pulso. Estos datos son suficientes para realizar la actualización de forma adecuada.

Siguiendo las bases del diseño de la memoria, los pasos necesarios para su completa gestión se muestran en forma general en la Fig. 4.11.

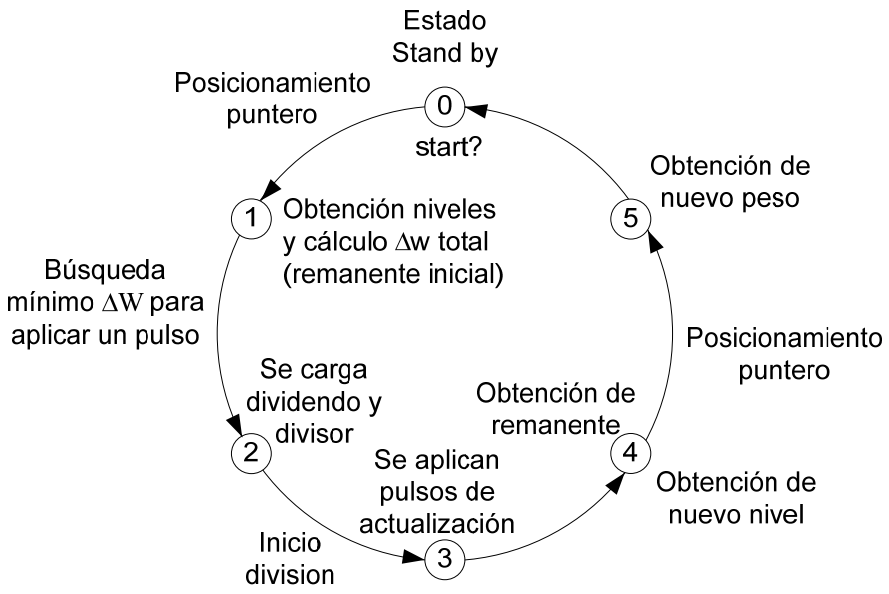


Fig. 4.11: Máquina de estados memoria

Las operaciones que consigue esta máquina de estados se especifican en los puntos siguientes:

1. Se espera el arranque para actualización por parte del módulo de jerarquía superior (sinapsis/perceptrón).
2. Memoria de pesos entrega en su puerto de salida la cantidad de niveles; también se obtiene paralelamente la actualización total, correspondiente al Δw aplicado más el remanente inicial, igualmente se apunta en memoria para recolectar el mínimo Δw para generar un pulso.
3. Memoria de pesos entrega mínimo Δw, se cargan los valores de dividendo y divisor y se inicia la división.
4. Se espera el resultado de la división y luego se calculan los pulsos de actualización como la parte entera del cuociente.
5. Se obtienen luego de los pulsos, el nuevo nivel a aplicar y el remanente sobrante.
6. Con el nuevo nivel se apunta en memoria de pesos y se rescata nuevo peso actualizado.

Con el diagrama de estados de la Fig. 4.11 y los pasos anteriormente enunciados, se tiene un

acercamiento más acabado del objetivo que representa la memoria en el proyecto.

Con estos principios declarados, resta indicar el diseño pertinente al divisor, encargado de calcular la cantidad de pulsos para actualizar la celda de memoria, los que se exponen a continuación.

4.3.5.1. Divisor

Las operaciones de división siempre ha sido provocación de malestar cuando ha sido obstáculo ante una implementación en FPGA. A diferencia de la suma, la resta, o la multiplicación, no hay operación lógica simple que genere un cuociente. Entre otras dificultades, la división difiere de otras rutinas de aritmética en que operaciones de punto flotante no producen un resultado de punto flotante predecible y finito. Hay, sin embargo, un número de formas para tratar este asunto; en este trabajo, se desarrolla una alternativa que genera resultados precisos y tarda su ejecución proporcionalmente a la cantidad de bits que se requieran dividir, el esquema de funcionamiento se muestra en la Fig. 4.12.

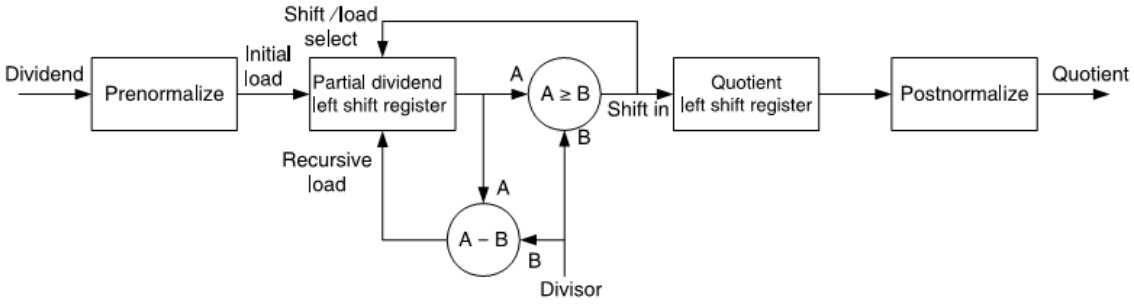


Fig. 4.12: Arquitectura de divisor

En términos generales, la operación que se lleva a cabo en el divisor consta de:

- Una pre-normalización del dividendo para reducir la cantidad de ciclos de reloj de ejecución.
- Desplazamiento del dividendo según sea el caso: si el dividendo (A) es mayor o igual que el divisor (B), se carga en el dividendo la resta de ambos y se desplaza a la izquierda (un bit), de lo contrario solo se desplaza el dividendo a la izquierda (un bit).

- Generación de cuociente, dependiendo de los resultados iterativos de la comparación del dividendo con el divisor, se van generando los bits de resultado que en cada iteración se desplaza el registro que los almacena a la izquierda (un bit), almacenando el nuevo resultado en su bit menos significativo.
- Luego de terminar la operación se post-normaliza el resultado, esto debido a pre-normalización que se lleva a cabo al inicio de la operación.

El diseño de la máquina de estados para llevar a cabo esta función se muestra en la Fig. 4.13.

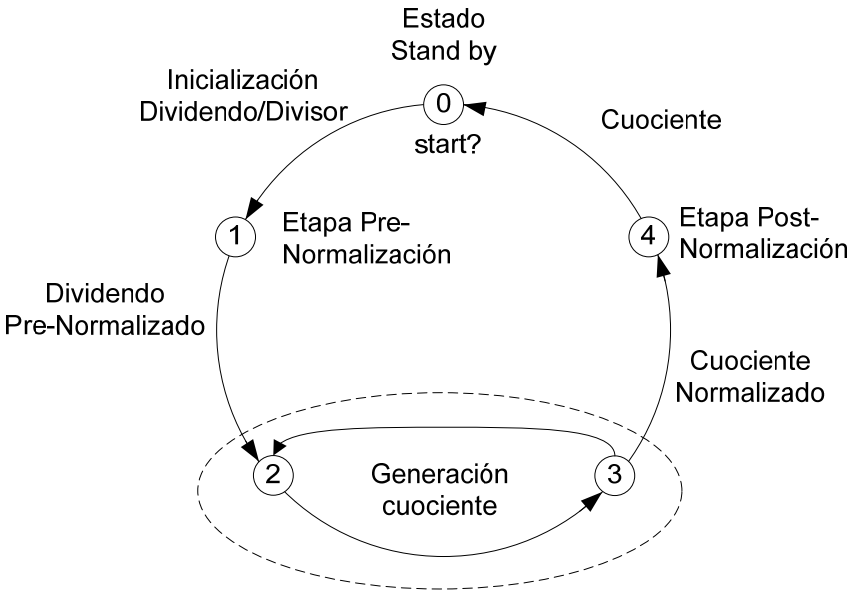


Fig. 4.13: Máquina de estados divisor

Con el desarrollo de los principios de diseño descubiertos, resta conocer las técnicas usadas para la implementación en lenguaje Verilog HDL de la descripción de estos módulos.

4.4. Implementación en FPGA

La implementación obedece a la impregnación de todos los principios vistos en las etapas de arquitectura y diseño, para el funcionamiento bajo cierto lenguaje y tecnología.

Primero que todo se introducen consideraciones generales para todos los diseños en la sección 4.4.1. En las secciones posteriores se muestra las consideraciones de mayor trascendencia para el desarrollo de las diferentes funciones; iniciando la sección 4.4.2 con la red neuronal,

posteriormente se especifican las observaciones del perceptrón en la sección 4.4.3, se revisa además en la sección 4.4.5, las premisas aplicadas al multiplicador. Finalmente se detallan los aspectos de mayor importancia para la celda de memoria en la sección 4.4.6.

4.4.1. Generalidades

Para evitar redundancia en el reconocimiento de módulos y variables. Se define una convención utilizada para la declaración de cada módulo en donde las entradas se declaran primero, posteriormente las salidas.

Las señales de reloj (clk) son siempre declaraciones de entrada, al igual que la señal de arranque (start).

Por otro lado, la señal que indica que el dispositivo se encuentra en operación o en reposo (ready), siempre será un puerto de salida, a excepción del manejo de señales internamente en la manipulación de módulos de jerarquía menores.

4.4.2. Red Neuronal

La función de esta unidad corresponde a la comparación de los valores de referencia dados por los multiplicadores de pesos fijos con el perceptrón adaptivo, para calcular su error y ser aplicado con la tasa de aprendizaje para el posterior cálculo. Grandes bloques de este componente son generadas por script de Matlab® (Anexo A.1.g). A continuación se muestran las porciones de código que identifican esta función.

La declaración del módulo depende de la cantidad de sinapsis a utilizar, en el ejemplo de se muestra, se declara un módulo con 2 perceptrones, en donde recibe dos elementos del vector de entrada.

```
module red_neuronal(x1, x2, start, clk, ready, error, new_weight1, oldremainder1, new_weight2,
oldremainder2);
```

En este bloque, los elementos *x1* y *x2* son las entradas que reciben componentes del vector de entrada *x*, por otro lado, los elementos *new_weigth1* y *new_weigth2* son salidas que no tienen otra función más que mostrar el valor actual de cada celda de memoria, también ocurre con *oldremainder1* y *oldremainder2*, los cuales muestran las cantidades de los remanentes presentes en

cada celda de memoria, el error va mostrando la evolución del acercamiento de los pesos de la red adaptiva a la fija.

La declaración del perceptrón conectado también está sujeto a la modificación mediante script de Matlab®, la línea que integra el perceptrón en la red se muestra a continuación.

```
perceptron per1(x1, dw1, x2, dw2, clk, startmults, startmems, readymems, readymults, Y, new_weight1,
oldremainder1, new_weight2, oldremainder2);
```

Las variables *dw1* y *dw2* son las actualizaciones a aplicar a cada sinapsis, *startmults* y *startmems* activan el arranque de los multiplicadores y las memorias en todas las sinapsis presentes, Y es la salida del perceptrón, correspondiente a la suma de las salidas de todas las sinapsis.

Antes de iniciar las operaciones de entrenamiento de los pesos, es de vitalidad inicializar estos y cargar los remanentes iniciales en 0. Para esto se activa el actualizador de memoria al inicio de la ejecución de en la FPGA.

```
initial begin                    //Inicializa las memorias para cargar los valores iniciales
        startmems <= 1;
end
```

Como se han declarado dos sinapsis para este ejemplo, se requieren declarar dos multiplicadores de pesos fijos, los cuales se muestran en el bloque siguiente.

```
multiplicador2 #(.nmult(44)) m1(startm1, clk, x1, 16'hF33B, wx1, readym1);
multiplicador2 #(.nmult(16)) m2(startm2, clk, x2, 16'h3B2C, wx2, readym2);
```

En este caso, los pesos son iniciados directamente, se puede apreciar además que multiplicadores son utilizados (multiplicadores 44 y 16), las señales *wx1* y *wx2* corresponden a las multiplicaciones con el vector de entrada *x*.

Luego de tener las declaraciones, se procede a mostrar piezas clave del funcionamiento de la red neuronal.

El estado inicial arranca las multiplicaciones tanto del perceptrón fijo como en el adaptivo; en el estado siguiente, se espera por los resultados del perceptrón de pesos fijos para calcular la referencia *d*.

```
if (readym1 && ~readymu_1_up) begin
        mul_sinapsis_counter <= mul_sinapsis_counter - 1;
        d <= d + wx1;
        readymu_1_up <= 1;
end
else if (readym2 && ~readymu_2_up) begin
        mul_sinapsis_counter <= mul_sinapsis_counter - 1;
        d <= d + wx2;
        readymu_2_up <= 1;
end
if (mul_sinapsis_counter == 0) state <= state + 1;
```

La realización espera tanto a la sinapsis 1 como a la segunda para ir calculando la referencia. Luego se procede a calcular el error, para esto se requiere que el perceptrón adaptivo entregue su señal de finalización.

```
if (readymults) begin //si se han hecho las multiplicaciones, Y contiene W'*x
        error <= d - Y;
        state <= state + 1;
end
```

Finalmente, se muestra la actualización de los pesos,

```
dw1 <= delta1_final;
dw2 <= delta2_final;

startmems <= 1;
state <= state + 1;
```

Las variables *delta1_final* y *delta2_final* poseen la actualización según el algoritmo LMS; es decir, la multiplicación del error con el vector de entrada *x* y la tasa de aprendizaje, el cuál se fija en 1/64, este valor se fija de manera arbitraria y busca que la convergencia de pesos se realice de la forma más rápida, pero sin oscilaciones de magnitud considerable.

Cabe destacar que en este módulo, se pueden implementar distintos algoritmos de redes neuronales, que tengan como base un perceptrón lineal.

4.4.3. Perceptrón

El perceptrón adaptivo conecta las sinapsis para calcular la salida correspondiente a la suma de cada multiplicador, para esto requiere invocar cada sinapsis a utilizar,

```
//Modulos sinapsis presentes -- X
sinapsis #(.numero(44),.initial_index(223)) synapse1(dw1, x1, clk, start_mult1, start_mem1, y1,
ready_mem1, ready_mult1, new_weight1, oldremainder1);

sinapsis #(.numero(16),.initial_index(125)) synapse2(dw2, x2, clk, start_mult2, start_mem2, y2,
ready_mem2, ready_mult2, new_weight2, oldremainder2);
```

de aqui se rescatan las variables *y1* y *y2*, que corresponden a las salidas de cada multiplicador, se observa además de estas declaraciones, la iniciación de pesos, estos se indican como niveles en cada celda de memoria, en donde la sinapsis que utiliza los datos de la celda de memoria y multiplicador 44, es iniciado en 223, este valor depende de la cantidad de niveles disponibles en cada celda, por ejemplo, los niveles de la celda 44 es de 367, por lo tanto 223 esta cerca de la mitad hacia arriba, por lo tanto el valor del peso está por sobre los 0 volts. En la otra celda (16), el indice es 125, la cantidad de niveles para esta celda es de 256, por lo tanto el peso almacenado esta por cerca de los 0 volts por debajo.

La adquisición de la salida del perceptrón es similar a la obtención de la referencia en la red neuronal,

```
if (ready_mult1 && ~readymu_1_up) begin
mul_sinapsis_counter <= mul_sinapsis_counter - 1;
        Y <= Y + y1;
        readymu_1_up <= 1;
end
else if (ready_mult2 && ~readymu_2_up) begin
        mul_sinapsis_counter <= mul_sinapsis_counter - 1;
        Y <= Y + y2;
        readymu_2_up <= 1;
end
if (mul_sinapsis_counter == 0) mulstate <= 0;
```

La actualización de la salida *Y* no puede ser llevada a cabo simultáneamente por cada sinapsis, es por esto que se decide implementar esta técnica, que espera que las sinapsis entreguen sus componentes, y a medida que van levantando su señal *ready*, van actualizando la salida para luego bloquearse y no ejecutarse más, y el contador de multiplicadores se decrementa, cuando esta llega a 0, se puede proceder a finiquitar la operación.

Si se observa el diseño del perceptrón en 4.3.2, se notará que también maneja las celdas de memoria de cada sinapsis, la forma en que hace esto es esperando que cada sinapsis indique que su celda de memoria correspondiente termina su actualización, para luego entregar a la red neuronal la señal que se finiquitó la actualización de todos los pesos en todas las sinapsis, la forma de proceder es similar a las antes vistas cuando se manejan varios flag de estado ocioso (*ready*).

```
if (ready_mem1 && ~readyme_1_up) begin
        mem_sinapsis_counter <= mem_sinapsis_counter - 1;
        readyme_1_up <= 1;
end
else if (ready_mem2 && ~readyme_2_up) begin
        mem_sinapsis_counter <= mem_sinapsis_counter - 1;
        readyme_2_up <= 1;
end
if (mem_sinapsis_counter == 0) memstate <= 0;
```

4.4.4. Sinapsis

Como se describió con anterioridad, el módulo sinapsis agrupa la celda de memoria con el multiplicador análogo correspondiente, para esto invoca los módulos que implementan estas funciones.

```
//Modulos memoria y multiplicador
memoria #(.memory(numero)) memory(start_mem, clk, oldindex, deltaw, oldremainder, newremainder,
newindex, new_weight, ready_mem);

multiplicador #(.nmult(numero)) multtester(start_mult, clk, x, new_weight, y, ready_mult);
```

La sinapsis tiene un parámetro el cual es utilizado en el perceptrón, el cual indica cual celda y memoria utilizar, este parámetro es *numero*, y este es aplicado en la declaración de la memoria y multiplicador. En este modulo además se cargan los valores de peso inicial y remanente inicial en la memoria,

```
initial begin      //carga de indice de peso inicial en memoria
        oldindex <= initial_index;
        oldremainder <= 0;
end
```

Se finaliza la función de este módulo, actualizando los valores de remanente e índice de la celda de memoria, pudiéndose utilizar la nueva información para actualización futura de pesos.

```
always @(posedge ready_mem) begin
        oldindex <= newindex;
        oldremainder <= newremainder;
end
```

4.4.5. Multiplicador

La declaración inicial del multiplicador carece de variables, se especifican las necesarias, las cuales son:

```
module multiplicador(start, clk, x, w, product, ready);
        parameter nmult = 5;
```

La señal x proviene del universo de entrada, la cual multiplica al peso w, el resultado de la operación de multiplicar se entrega en *product*, y este tiene un valor admisible cuando ready permanece activo. El parámetro *nmult* indica cual dato de laboratorio utilizar para probar el desempeño.

El multiplicador declara para el cálculo aproximativo de la tangente hiperbólica, la memoria de parámetros m y b_y,

```
mem_m tanh_m_b(indextanh, clk, Out_m, Out_b);
```

La variable *indextanh* le indica a la memoria la posición para extraer los parámetros y entregarlos en *Out_m* y *Out_b*. Esta memoria "ROM" es diseñada para almacenar 512 parámetros correspondientes al tramo positivo de tangente hiperbólica, esto gracias a que la función es impar, por lo tanto se puede almacenar datos con el doble de precisión. La elección de 512 parámetros se debe a que un bloque de memoria RAM en bloque para la Virtex 2 Pro es de 1024 x 18 bit, como los parámetros son de 17 bits, se aprovecha la capacidad de un bloque completo para su uso.

Además del módulo que almacena los parámetros de la ecuación de la recta, se tiene además el módulo de memoria de parámetros a, b y c; estos parámetros sin embargo, varían dependiendo del multiplicador a utilizar, por lo tanto la invocación de este se realiza dentro de un bloque *generate*,

```
generate
        case (nmult)
        1: memabc #(.size(185), .filepath("data_mult1.txt")) memWabc(index, clk, abc_memOut);
        2: memabc #(.size(165), .filepath("data_mult2.txt")) memWabc(index, clk, abc_memOut);
                                                .
                                                .
                                                .
        64: memabc #(.size(177), .filepath("data_mult64.txt")) memWabc(index, clk, abc_memOut);
        endcase
endgenerate
```

La condición para seleccionar una de estas memorias, depende del parámetro nmult indicado con anterioridad. El bloque consta de 64 entradas, los cuales son creados con script de Matlab® (Anexo A.1.f), *filepath* indica el archivo a cargar y size el tamaño requerido para almacenar todos los parámetros. La organización de esta memoria se muestra en la Tabla 4.1.

#niveles			
w_1	a_1	b_1	c_1
w_2	a_2	b_2	c_2
.	.	.	.
.	.	.	.
.	.	.	.
$w_{niveles}$	$a_{niveles}$	$b_{niveles}$	$c_{niveles}$

Tabla 4.1: Organización memoria parámetros a, b y c

Siguiendo con el estudio de la implementación de la función multiplicadora, se revisa la sección de la máquina de estados que busca el peso más cercano al w ingresado para multiplicar.

```
if (dist < dist_winner) begin                    //si distancia se achica se sigue iterando
        dist_winner <= dist;                             //se asigna nueva distancia
        w_counter <= w_counter + 1;              //se incrementa contador de pesos

        if ( (w_counter + 1) > barridos) begin   //si se ha sobrepasado la cantidad de pesos --> se
                                                 //ha encontrado el peso
                index <= index + 1;              //se posiciona en constante a
                state <= state + 1;              //siguiente estado
        end
        else begin
                index <= index + 4;              //siguiente peso, no se ha llegado al final
                state <= 2;
        end
end
else begin                          //si la distancia aumenta es por que se ha pasado el peso mas cercano
        index <= index - 3;         //se posiciona en constante a
        state <= state + 1;         //se siguen extrayendo parametros a, b y c
end
```

La búsqueda se inicia calculando en un estado previo la distancia, esta se obtiene de la diferencia del peso ingresado a multiplicar, con los pesos almacenados en memoria. Luego se compara la distancia con la menor de las distancias (*dist_winner*), cuando la distancia calculada es mayor que la distancia menor, significa que el peso anteriormente evaluado se aproxima más que el peso utilizado en la iteración presente, por lo tanto el índice de memoria retrocede 3 espacios (se salta hacia los parámetros a, b y c). Si el peso adecuado es el último del multiplicador a utilizar, entonces la variable *dist_winner* siempre será asignada en cada iteración, para detectar cuando se ha llegado a esa instancia, se revisa que el contador de pesos revisados (*w_counter*) no supere la cantidad de niveles admitidos (representado por barridos).

Un aspecto que no se puede obviar de la implementación del multiplicador es el escalamiento de *by*, que representa el offset en la ecuación de la recta y también la constante *c*, que representa el offset en la ecuación que interpreta la curva de un peso *w*; ambos tienen la forma:

$$z = \alpha \cdot \Phi + \beta \tag{4.6}$$

Donde α y β pertenecen a la misma dimensión (β es el offset) y Φ es la variable de dimensión diferente; la diferencia de dimensiones produce una tercera dimensión al multiplicar dos variables de dimensiones dispares (α con Φ). Entonces β realiza un offset sobre la dimensión dominada por α y β, pero no incluye a Φ, para solucionar este problema, se requiere encontrar el valor "1" de Φ y aplicárselo a β mediante una multiplicación.

Ejemplo, si α y β son variables con valores que van desde 1 a 1024, en donde el "1" se asume en 128, y Φ es una variable con valores que van de 1 a 64 en donde el "1" es 32 la ecuación (4.6) tendría que ser reescrita como:

$$z = \alpha \cdot \Phi + \beta \cdot 32$$

Con eso el resultado tendrá la consistencia necesaria y no se perderá información aportada por el offset.

Con esta base se buscan los valores asumidos como "1" en la ecuación de la recta en ese caso se busca el "1" de *bx*, y en la ecuación de la curva (4.1) se busca el "1" entregado por la

función tangente hiperbólica, los resultados se obtienen desde Matlab® y son declarados como constantes en el multiplicador.

```
wire signed [17:0] constante2 = 18'h12C70;
wire signed [17:0] constante3 = 18'h12C6A;
```

La *constante2* es el ajuste para la curva que hace uso de la tangente hiperbólica, y la *constante3* es el ajuste para la ecuación de la recta.

4.4.6. Memoria

La declaración de puertos se muestra a continuación.

```
module memoria(start, clk, oldindex, change, oldremainder, newremainder, newindex, new_weight,
ready);
    parameter memory = 5;
```

Los puertos de entrada consisten en: *oldindex* quien recibe el nivel inicial de la celda de memoria, *change* es la actualización propiamente tal, *oldremainder* recibe el remanente resultante de una actualización anterior, en los puertos de salida se tienen: *newremainder* correspondiente al remanente quedante de la actualización, *newindex* indica el indice del nivel de memoria después de aplicar los pulsos de actualización; finalmente, *new_weight* entrega el valor del peso al cual fue actualizado, el parámetro *memory* indica que celda de memoria escoge el usuario para realizar las pruebas.

Este módulo posee similitud con el multiplicador, esta es debido a la generación de módulos de memoria; esta sin embargo crea instancias de memoria que almacena pesos sin parámetros *a*, *b y c*.

```
generate
        case (memory)
        1: memmem #(.size(48), .filepath("data_mem1.txt")) tablamemW(index, clk, memOut);
        2: memmem #(.size(43), .filepath("data_mem2.txt")) tablamemW(index, clk, memOut);
                                                .
                                                .
                                                .
        64: memmem #(.size(46), .filepath("data_mem64.txt")) tablamemW(index, clk,memOut);
        endcase
endgenerate
```

También son indicados en estos módulos, los nombres de archivos y el tamaño requerido para almacenar los pesos, la organización de esta memoria es como sigue.

#niveles
w_1
w_2
.
.
.
$w_{niveles}$
Δw promedio

Tabla 4.2: Organización memoria pesos

Con estos datos, se procede a detallar algunos de los bloques participantes en la actualización, uno de estos, obtiene el verdadero Δw, gracias a la adición del Δw aplicado más el remanente de la actualización anterior.

```
levels <= memOut[12:0];//se almacena cantidad de niveles de modulo de memoria escogido
index <= index + memOut[12:0] + 1;      //se posiciona puntero en delta promedio
realchange <= change + oldremainder;    //Se considera remanente anterior

state <= state + 1;
```

Se rescata además la extracción la cantidad de niveles presentes en aquella celda de memoria. Un detalle no menor consta de la precaución tomada cuando la memoria se encuentra en su máximo valor o mínimo valor, en donde se eliminan los remanentes, el código se muestra en el bloque siguiente.

```
if (signed_newindex < 1) begin  //si indice es menor que el minimo (1)
        newindex <= 1;
        index <= 1;                     //se apunta a posicion de memoria para obtener el peso
        newremainder <= 0;              //se resetea remanente por llegar a un limite
end
else if (signed_newindex > {1'b0,levels} ) begin //si indice supera cantidad de niveles
        newindex <= levels;
        index <= levels;
        newremainder <= 0;              //se resetea remanente por llegar a un limite
end
else begin                              //si indice se encuentra en rango aceptable
        newindex <= signed_newindex[12:0];
        index <= signed_newindex[12:0];
        newremainder <= realchange - extraccion_peso;
end

state <= state + 1;
```

Se aprecia de este código que el cálculo del remanente se realiza sólo cuando el índice de memoria se encuentra dentro del rango de valores permisibles, cuando se baja del mínimo o se supera el máximo, el remanente queda en 0 y el peso se mantiene en valor alto hasta que se produzca una tendencia a la baja. Se verifica además que el índice de memoria toma los mismos valores que el indicador de nuevo índice, esto con el fin de entregar en el siguiente periodo de reloj el nuevo valor del peso.

Ya se han revisado los aspectos relevantes de la memoria, se termina la etapa de implementación con el módulo divisor.

4.4.6.1. Divisor

Se inicia el detalle de la implementación declarando las variables requeridas para su correcta operación.

```
module divider(dividendo, divisor, clk, start, cuociente, ready);
        parameter decimal_bits = 8;
```

Las entradas como bien se pueden apreciar, corresponden al *dividendo* y *divisor*, luego que *ready* levanta su flag, se obtiene el resultado en *cuociente*, el parámetro *decimal_bits* indica cuantos bits de decimales contendrá cuociente al término de la división.

Las divisiones para efectos de desarrollarlas siempre entre números positivos, se observan los bits de signo de cada operando, para finalmente entregar la salida en función del XOR entre ambos signos.

```
wire cuociente_sign = dividendo_sign ^ divisor_sign;
assign cuociente = cuociente_sign ? -cuociente_unsig : cuociente_unsig;
```

Un aspecto de relevancia en la división, consiste en la etapa de pre-normalización, esta define la forma de operar en el resto de la máquina de estados. Para efectos de poder hacer divisiones entre operandos de dimensiones completamente diferentes; es decir, entre dividendo y divisor de varios órdenes de magnitud de distancia; se definen dos registros de 18 bits para manejar los movimientos de datos, estos son *p_decimal* y *p_entera*. Como bien indican sus nombres, *p_entera* contiene la parte entera del dividendo y p_decimal la parte decimal, como ambos registros son de largo considerable, las divisiones que se logran realizar pueden contener gran exactitud cuando los operandos son muy diferentes. En el siguiente codigo se muestra los estados encargados de realizar el desplazamiento de bits para lograr tener un dividendo que sea menor al doble del divisor.

```
1: begin                                        //Etapa de Prenormalizacion (Dividendo < 2 * Divisor)
        if ( p_entera >= divisor_double ) begin
                if (p_entera[0] == 1) p_decimal <= {1'b1,p_decimal[17:1]};
                else p_decimal <= {1'b0, p_decimal[17:1]};

                state <= 5;
                c <= c + 1;
        end
        else begin
                state <= state + 1;
        end
end
5: begin
        p_entera <= {1'b0, p_entera[17:1]};
        state <= 1;
end
```

El estado extra se define así por los conflictos que se podrían generarse cuando se requieren hacer lecturas y escrituras con asignaciones no-bloqueantes.

Con estas implementaciones, se prosigue a presentar las simulaciones de estos bloques para posteriormente transferirlos estos a la FPGA para probar el funcionamiento en tiempo real.

4.5. Conclusiones

En este capítulo se ha descrito todas las consideraciones de relevancia en el diseño, pasando por la revisión de las arquitecturas y terminando con las implementaciones en lenguaje Verilog HDL. Se observa en el transcurso del diseño mismo que el multiplicador que representa el forward path, tiene muchas observaciones con respecto a los periodos de reloj que requiere para obtener una salida representativa de una multiplicación con un multiplicador análogo incrustado en VLSI. Los parámetros de la curva *a*, *b* y c, se acceden en distintos periodos de reloj, se recomienda para mejorar las características de rendimiento, almacenar los parámetros en memorias independientes para su obtención más expedita.

En el emulador no es considerada la construcción incluyendo ruido Gaussiano, esto se debe a que se asume que el emulador cuando interactúe con dispositivos externos a este, recibirá información que tendrá cierto movimiento correspondiente a la captura del dato, lo que se puede dar fácilmente al momento de realizar la conversión análoga a digital si es que el emulador está conectado a un circuito análogo.

La tasa de aprendizaje se mantiene en un valor constante en las consideraciones iniciales, esto no quiere decir que pueda ser mejorado para emular diferentes algoritmos de aprendizaje en el futuro.

En el siguiente capítulo se muestran las simulaciones obtenidas para los bloques desarrollados y expuestos en las secciones antes vistas.

Capítulo 5. Simulaciones

5.1. Introducción

Utilizando los módulos descritos en el capítulo anterior, se realizaron distintas pruebas, pasando por el funcionamiento de cada brazo del emulador, para culminar probando el filtro lineal adaptivo de la sección 2.2.1, incorporando los efectos de los multiplicadores análogos en el forward path y de las memorias análogas en el feedback path. Las características de transferencia de estos dispositivos fueron obtenidas a partir de datos de laboratorio facilitados por el profesor Dr. Miguel Figueroa. Estos corresponden a mediciones realizadas a un arreglo de 64 multiplicadores que integran memorias análogas. Las simulaciones son categorizadas en 4 apartados. Se presenta inicialmente los resultados de simulación del multiplicador en la sección 5.2, posteriormente en la sección 5.3, se simula el comportamiento a esperar de una celda de memoria, se sigue con los ensayos para una sinapsis que integre ambos módulos descritos en la sección 5.4, finalizando con las simulaciones de la constitución de una pequeña red consistente en un perceptrón en la sección 5.5.

5.2. Multiplicadores análogos

La forma de probar el multiplicador o cualquier módulo desarrollado es haciendo pruebas robustas para asegurar el buen funcionamiento, este paso es de vital importancia, ya que el emulador del multiplicador debe entregar para toda entrada y peso, la misma respuesta que entregaría un multiplicador análogo implementado en VLSI. Los ensayos realizados se dividen en dos secciones, se inicia la ensayo de la construcción de la función tangente hiperbólica mediante rectas en las sección 5.2.1, posteriormente se termina observando las características de transferencia originales y las entregadas por simulación en la sección 5.2.2.

5.2.1. Aproximación de tangente hiperbólica

Como bien se ha expuesto en el capítulo anterior, la tangente hiperbólica a utilizar se basa en el desarrollo de rectas que barren el rango de valores a utilizar por los multiplicadores, para probar su correcta síntesis antes de ser implementado en FPGA, se hacen pruebas en ambiente Matlab®; en la Fig. 5.1, se muestran los datos de laboratorio para el multiplicador 28,

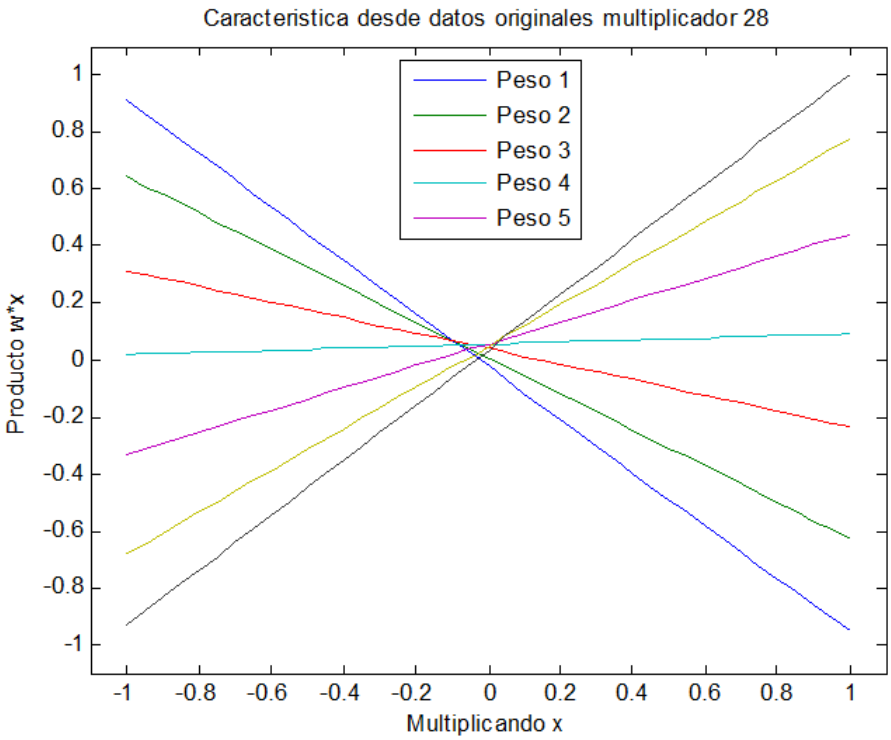


Fig. 5.1: Característica datos originales, multiplicador 28

Los datos de laboratorio ofrecidos, tienen características similares debido al rango reducido del cual fueron tomadas las muestras, esto con el fin de eliminar en lo posible las saturaciones de la tanh para valores grandes, pero aún así mantienen la forma de tanh para valores disminuidos y contienen el efecto del *device mismatch*. Se revisa ahora la función tanh de Matlab®, utilizando los parámetros *a, b y c* se busca conseguir una característica de transferencia de propiedades similares.

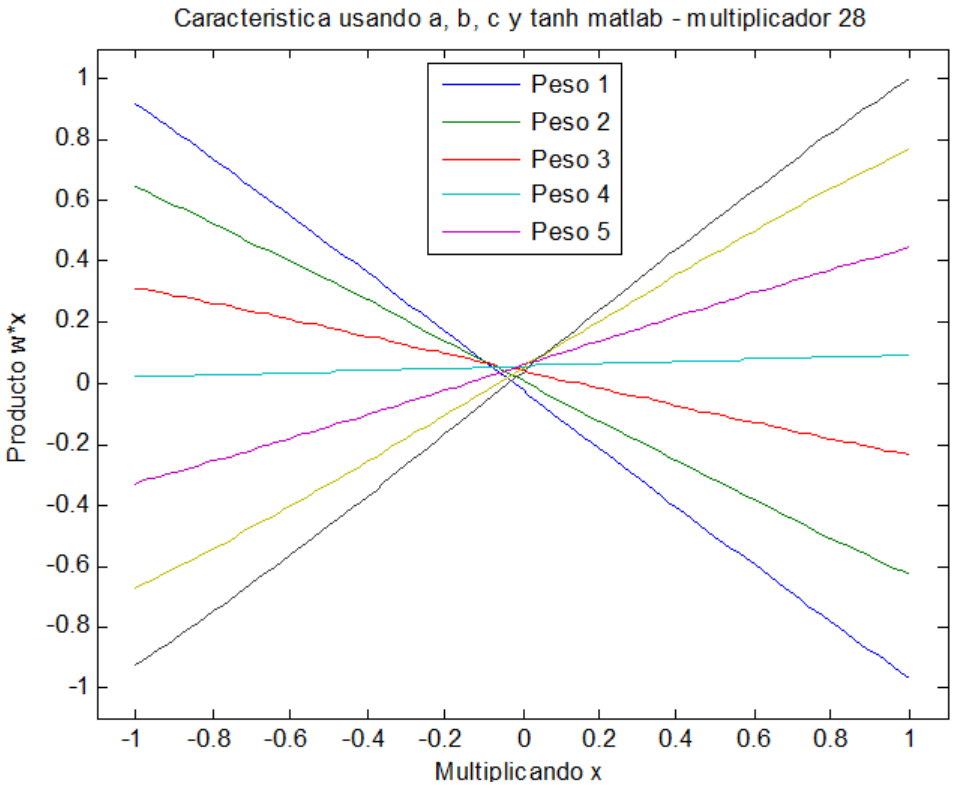


Fig. 5.2: Característica de transferencia multiplicador 28, con tanh Matlab®

La tangente hiperbólica a utilizar contiene 512 ecuaciones de la recta, estas rectas son adecuadas en rango al espectro de datos de laboratorio; es decir, se revisa en los 64 multiplicadores los valores que toma el argumento de tanh, esto por medio del parámetro *b*, el máximo *b* generará el mayor *bx*, con eso se asegura la utilización al 100% de la aproximación por rectas de la función tanh; en la Fig. 5.3, se presenta el tramo a utilizar para la construcción de los parámetros de las rectas.

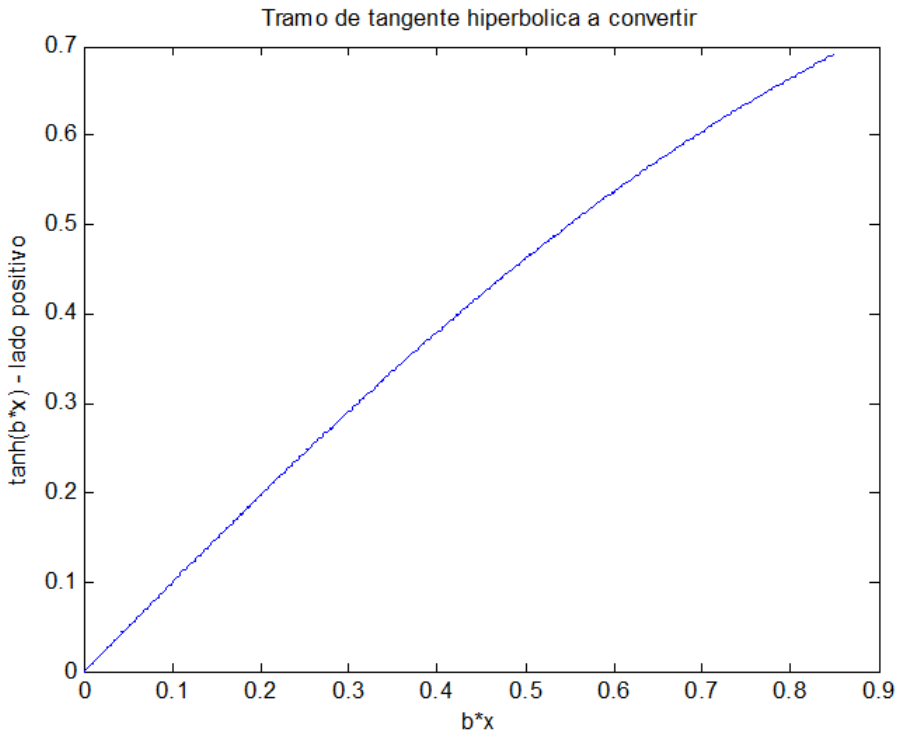


Fig. 5.3: Tramo tangente hiperbólica para almacenar las rectas en memoria

Se prueban estos parámetros, haciendo un barrido de *bx*, desde valores negativos hasta el máximo bx positivo, los resultados se muestran en la Fig. 5.4.

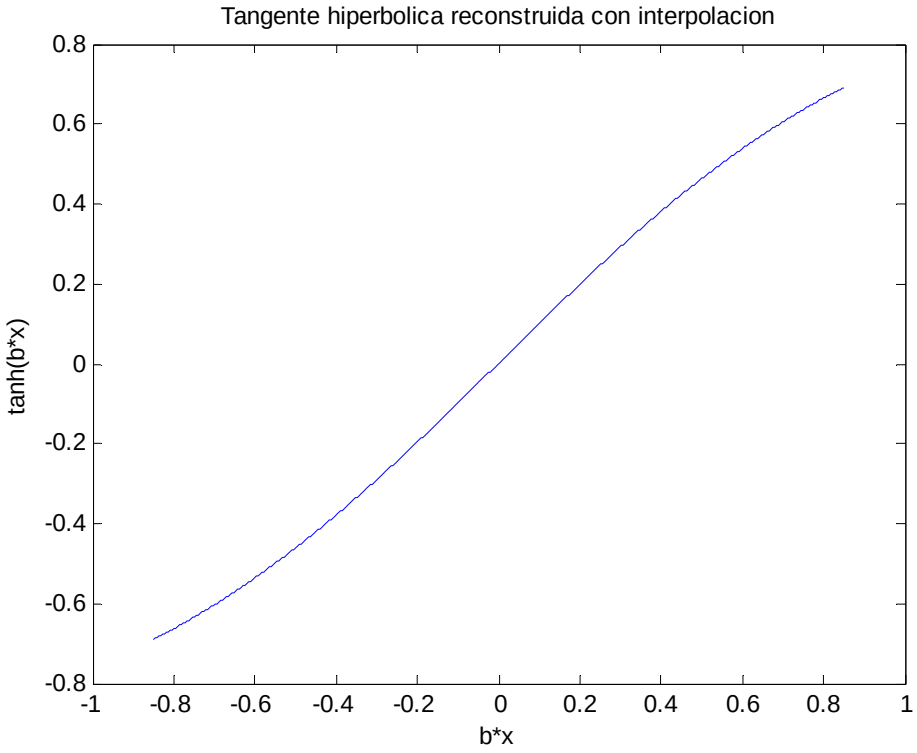


Fig. 5.4: Reconstrucción tanh con interpolación

Se verifica entonces que los 512 parámetros pueden entregar resultados satisfactorios para su utilización en representación de la tanh en la FPGA. Ahora se verifica que el barrido de multiplicaciones usando esta tanh entrega resultados similares a los vistos usando tanh de Matlab® y los datos originales de laboratorio, se repite el ensayo utilizando el multiplicador 28, el cual se puede apreciar en la Fig. 5.5.

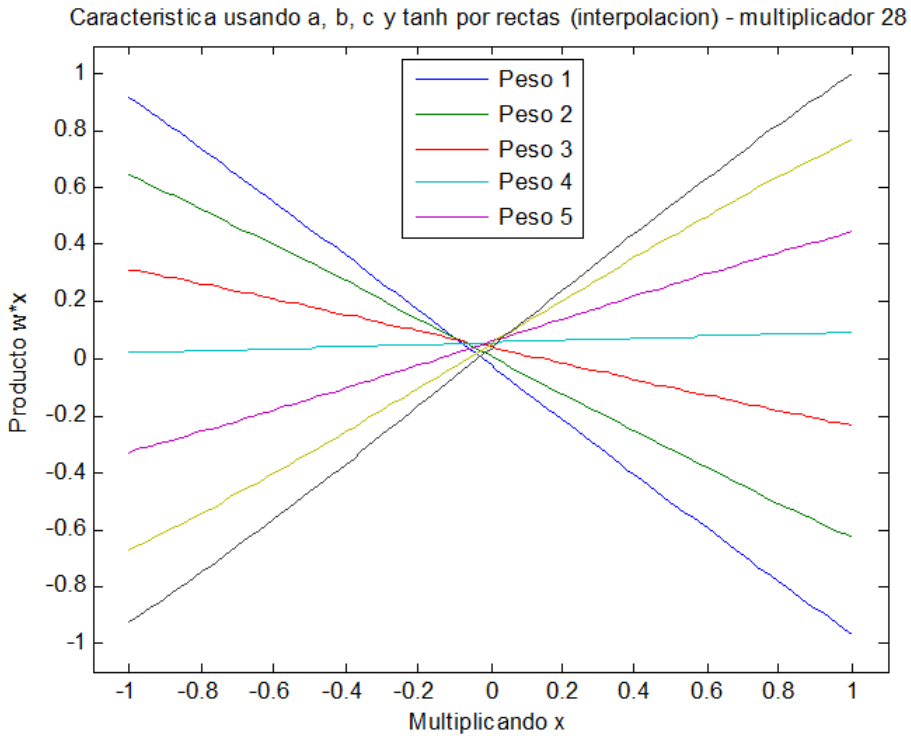
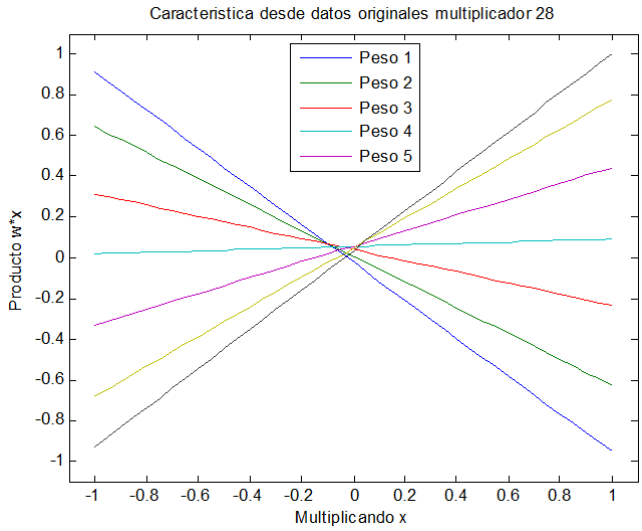


Fig. 5.5: Característica multiplicador 28 - usando tanh con rectas

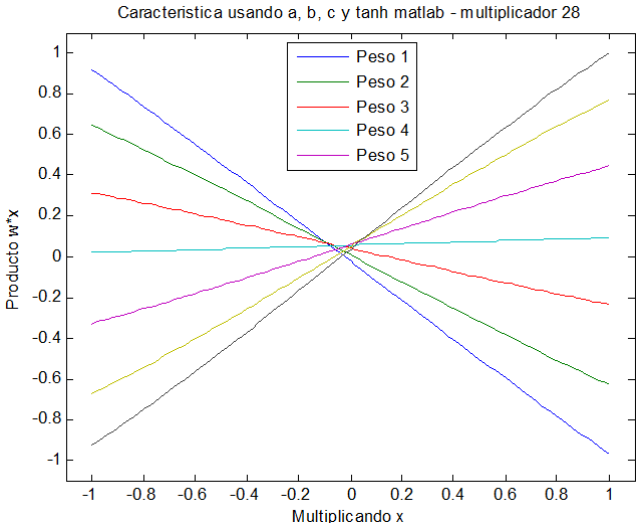
Como se aprecia en las figuras (Fig. 5.1, Fig. 5.2 y Fig. 5.5), las formas de los barridos de multiplicaciones tienen formas similares, los datos de laboratorio se diferencia un poco de las gráficas usando tanh debido a que esta no sigue fielmente la función matemática, pero los barridos utilizando los parámetros *a*, *b* y *c* (Fig. 5.2 y Fig. 5.5) poseen una similitud superior, con lo cual se finaliza las pruebas y se concluye con la aprobación de la función para ser utilizada en reemplazo de tanh como función matemática.

5.2.2. Multiplicación

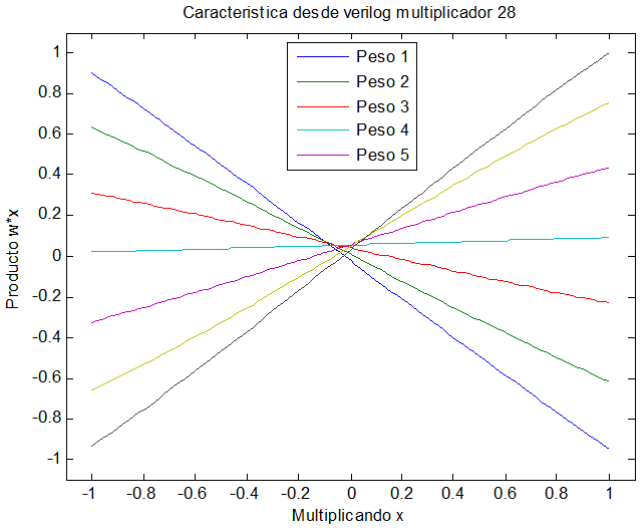
Los resultados antes vistos, utilizan los datos de laboratorio en Matlab® para desplegar los barridos (mediciones de laboratorio y constantes *a*, *b* y *c*). En esta oportunidad, los ensayos realizados son efectuados en el simulador de Xilinx® ISE™ Webpack™ para el módulo multiplicador, se programa un banco de pruebas (*test bench*) que guarda a archivo cada resultado del barrido de las multiplicaciones, esto con el fin de recopilar los datos en Matlab® y graficarlos para contrastarlos con los resultados originales. En la Fig. 5.6 se muestran gráficas del multiplicador simulado en Verilog HDL en oposición con los datos de laboratorio y tanh Matlab®.


Caracteristica desde datos originales multiplicador 28
Peso 1
Peso 2
Peso 3
Peso 4
Peso 5
Producto w*x
Multiplicando x

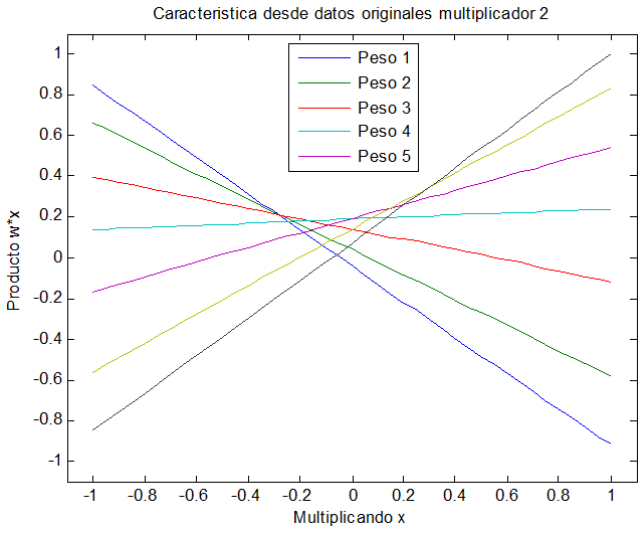
(a)


Caracteristica usando a, b, c y tanh matlab - multiplicador 28
Peso 1
Peso 2
Peso 3
Peso 4
Peso 5
Producto w*x
Multiplicando x

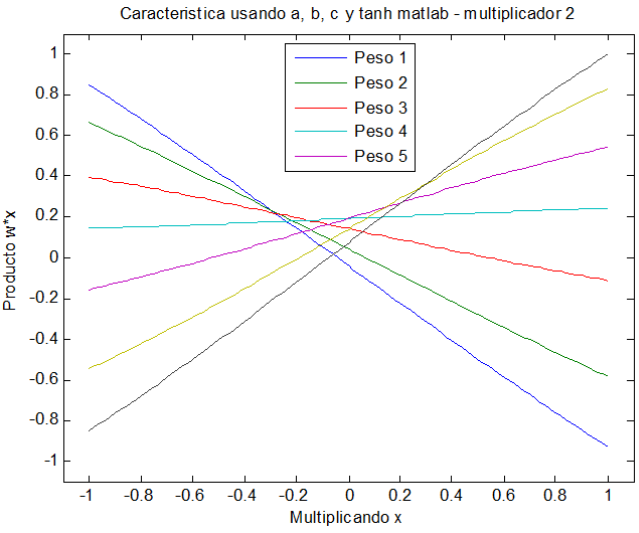
(b)


Caracteristica desde verilog multiplicador 28
Peso 1
Peso 2
Peso 3
Peso 4
Peso 5
Producto w*x
Multiplicando x

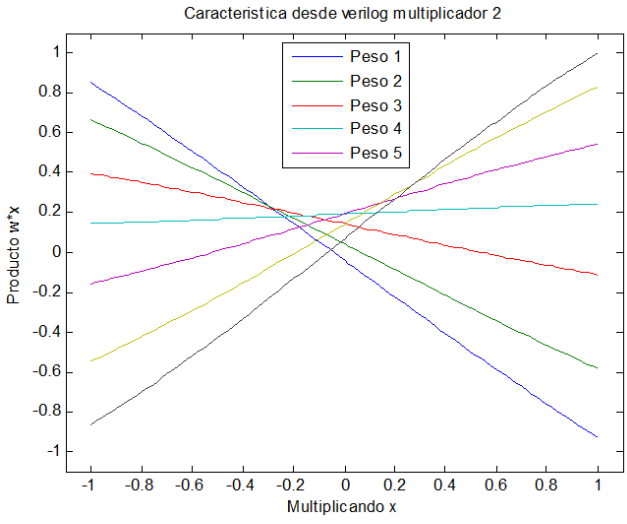
(c)


Caracteristica desde datos originales multiplicador 2
Peso 1
Peso 2
Peso 3
Peso 4
Peso 5
Producto w*x
Multiplicando x

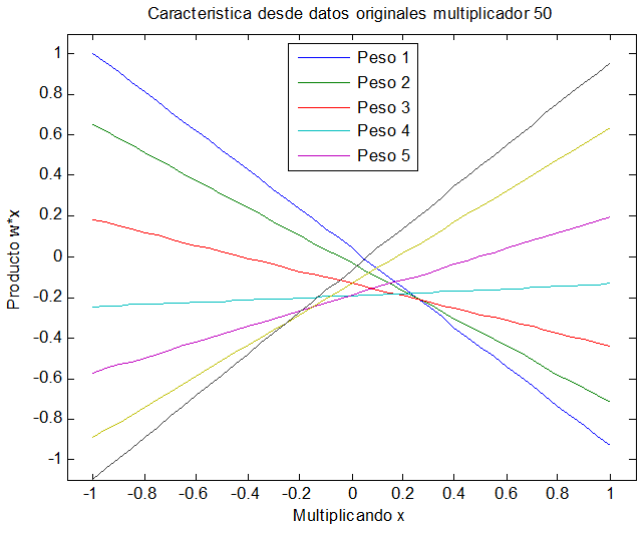
(d)


Caracteristica usando a, b, c y tanh matlab - multiplicador 2
Peso 1
Peso 2
Peso 3
Peso 4
Peso 5
Producto w*x
Multiplicando x

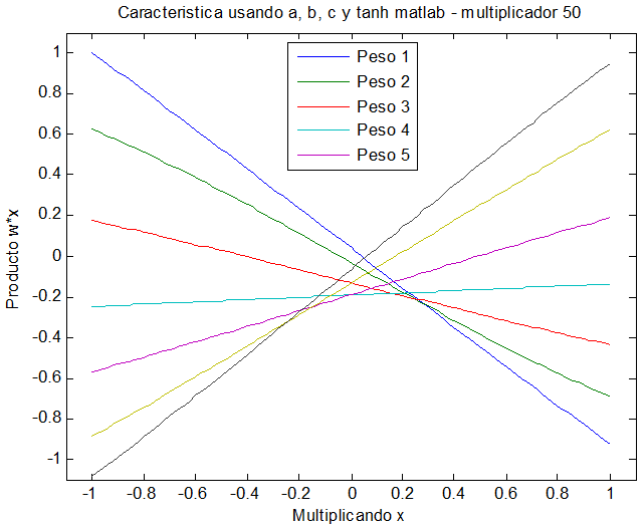
(e)


Caracteristica desde verilog multiplicador 2
Peso 1
Peso 2
Peso 3
Peso 4
Peso 5
Producto w*x
Multiplicando x

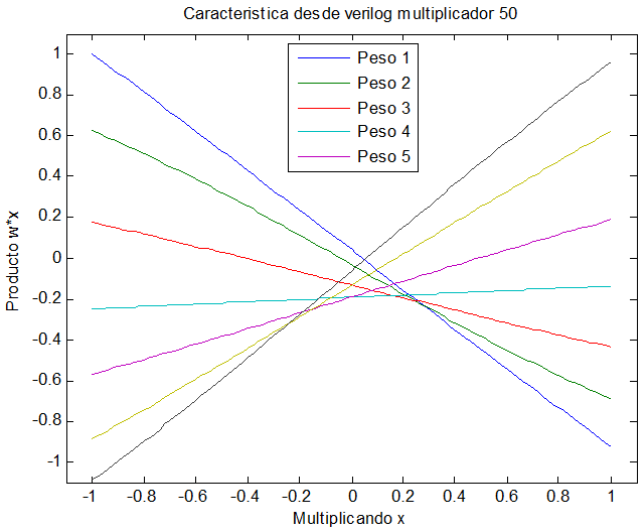
(f)



(g)



(h)



(i)

Fig. 5.6: Multiplicaciones con datos de: Laboratorio, tanh Matlab®, módulo en Verilog HDL.

(**a**), (**b**) y (**c**) multiplicador 28. (**d**), (**e**) y (**f**) multiplicador 2. (**g**), (**h**) y (**i**) multiplicador 50.

Se concluye la correcta operación llevada a cabo en la emulación del multiplicador, en donde para diferentes datos de laboratorio, el emulador se comporta adaptando sus parámetros y entregando resultados altamente equivalentes.

5.3. Celdas de memoria

El módulo de la celda de memoria, no requiere hacer grandes cálculos como ocurre en el multiplicador, esto gracias al divisor que simplifica enormemente la obtención rápida y precisa de un cuociente para la obtención de la cantidad de pulsos. Es por este motivo que se presentan resultados de algunas operaciones a las cuales puede verse recurrido la celda de memoria y no se hacen barridos como se hace en el multiplicador. Los ensayos se realizan directamente en el simulador de Xilinx Webpack™, los cuales se pueden apreciar en la Fig. 5.7.

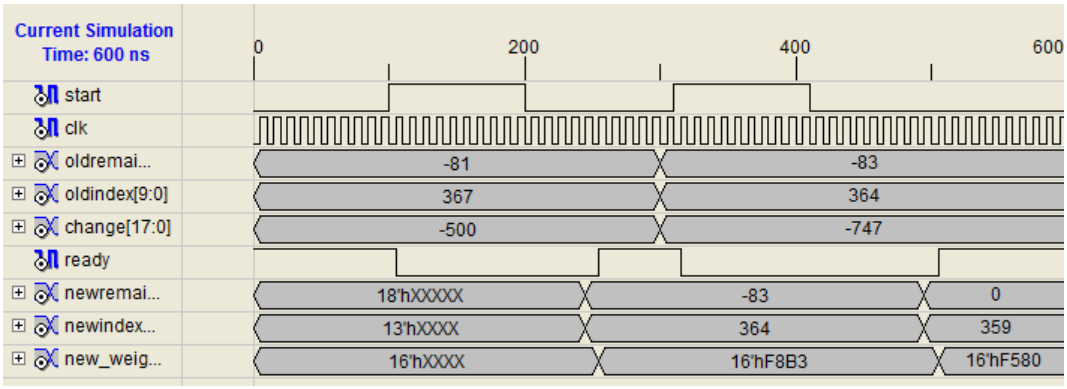


Fig. 5.7: Simulación módulo memoria en Verilog HDL.

En esta simulación se aplican dos cambios para observar la respuesta de la memoria, en primer lugar se ingresan valores iniciales para la memoria numero 44 de laboratorio, esta celda de memoria contiene 367 niveles. En la Tabla 5.1 se muestran algunos de los datos contenidos en las últimas posiciones de memoria.

Nivel	Memoria (HEX)
359	**F580**
360	F620
361	F6CF
362	F764
363	F804
364	**F8B3**
365	F944
366	F9E4
367	**FA88**

Tabla 5.1: Últimas entradas de memoria 44

La última posición es en donde es iniciado el puntero de memoria (revisar *old_index*), el delta promedio de esta memoria es de 166, por lo tanto cada cambio que sea mayor que este valor, producirá un pulso, por ejemplo si se aplica una actualización de 170, se genera un pulso y un remanente de 4. En el caso de la simulación se aplica una actualización de -500, sumado a la supuesto remanente anterior de -81, producirán una actualización total de -581 (tres pulsos y medio), por lo tanto se retrocede el índice en 3 posiciones quedando en 364 y un remanente de -83 (medio pulso).

En la segunda prueba se mantienen los índices y valores anteriores para simular la actualización previa, y se aplica esta vez una actualización de -747 (equivalente a cuatro pulsos y medio), la memoria reúne esta actualización con el remanente anterior (medio pulso) y se genera un desplazamiento total de 5 pulsos, por lo tanto el índice queda en 359 y arroja el valor del peso como se aprecia en la ventana de simulación de la Fig. 5.7.

5.4. Sinapsis

El test de sinapsis refleja la integración del módulo de memoria y multiplicador en una misma celda (similar a Fig. 4.7). Una muestra de ello se observa en la ventana de simulación de las Fig. 5.8 y Fig. 5.9.

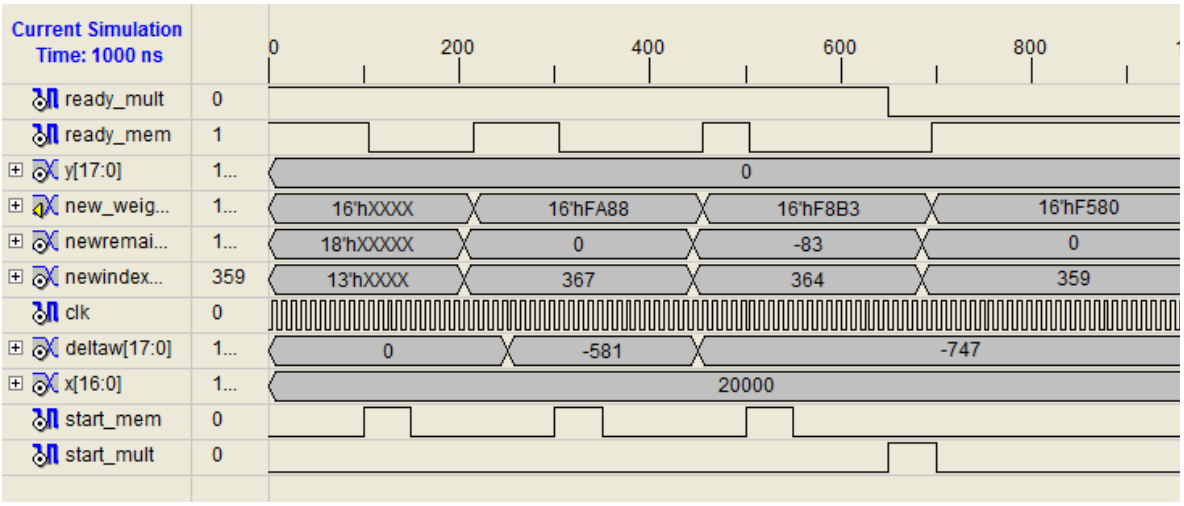


Fig. 5.8: Simulación sinapsis – zoom

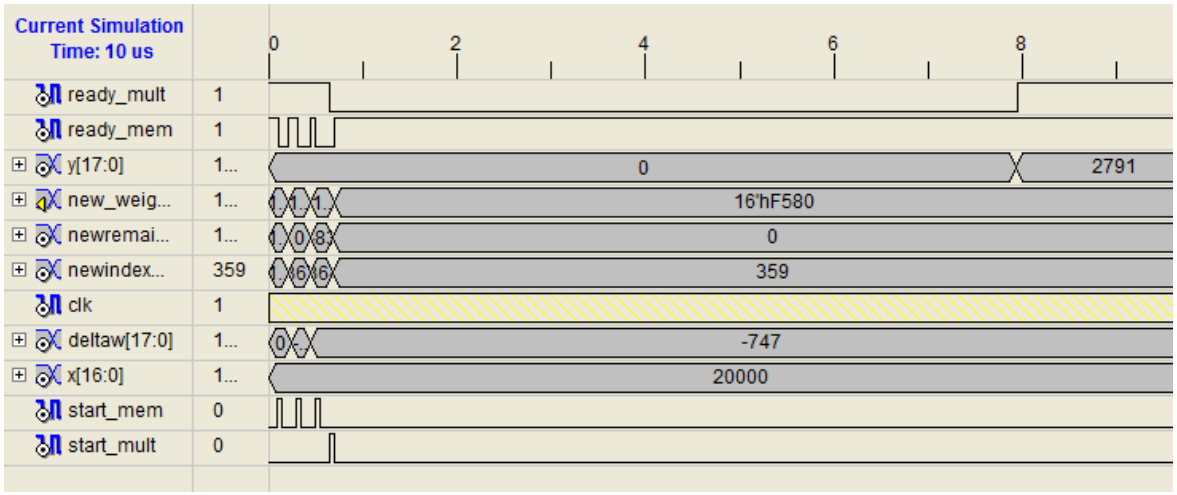


Fig. 5.9: Simulación sinapsis

De la Fig. 5.8, se aprecia como la memoria va actualizando su valor automáticamente, debido a la realimentación implícita otorgada en la sinapsis, después de aplicar dos actualizaciones, de la misma forma que se realizó en la simulación del módulo de memoria, se observan los mismos cambios vistos en 5.3. Posteriormente se aplica una multiplicación mediante la activación de la variable *start_mult*, resultando en la salida "*y*" con el valor correspondiente a la multiplicación del peso (16´hF580) con x (20000). Se verifica el funcionamiento adecuado en ambas situaciones.

5.5. Red neuronal

Se finalizan las simulaciones presentando el producto de los bloques base emulados, se implementa la simulación de una red neuronal con las características de la Fig. 4.1. Para esto se crea una red de pesos fijos y la red se encargara de calcular los errores que se vayan generando, al mismo tiempo que produce las actualizaciones. El ensayo a presentar es generado por script de Matlab® (Anexo A.1.g), en donde se ingresan las sinapsis con los datos de laboratorio a utilizar, un ejemplo del script ejecutándose en la línea de comando se muestra a continuación.

```
Ingrese la(s) sinapsis (de 1 a 64) para generacion de perceptron,
 si ingresa mas de una escriba dentro de []: [16 44 51 54 18]

Ingrese la cantidad de puntos a presentar a la red (ej. 1024): 1024
```

Luego de ingresar los datos solicitados, se genera el código Verilog HDL que crea la cantidad de sinapsis deseadas que emulan los datos de laboratorio escogidos.

Las sinapsis ingresadas para la prueba de la red corresponden a las ingresadas en el bloque anterior, la cantidad de puntos a utilizar corresponde a la cantidad de iteraciones que se presenta a la red un vector del universo de entrada con la particularidad de ser este vector de valores aleatorios.

La evolución del error medio cuadrático para esta simulación se observa en la Fig. 5.10.

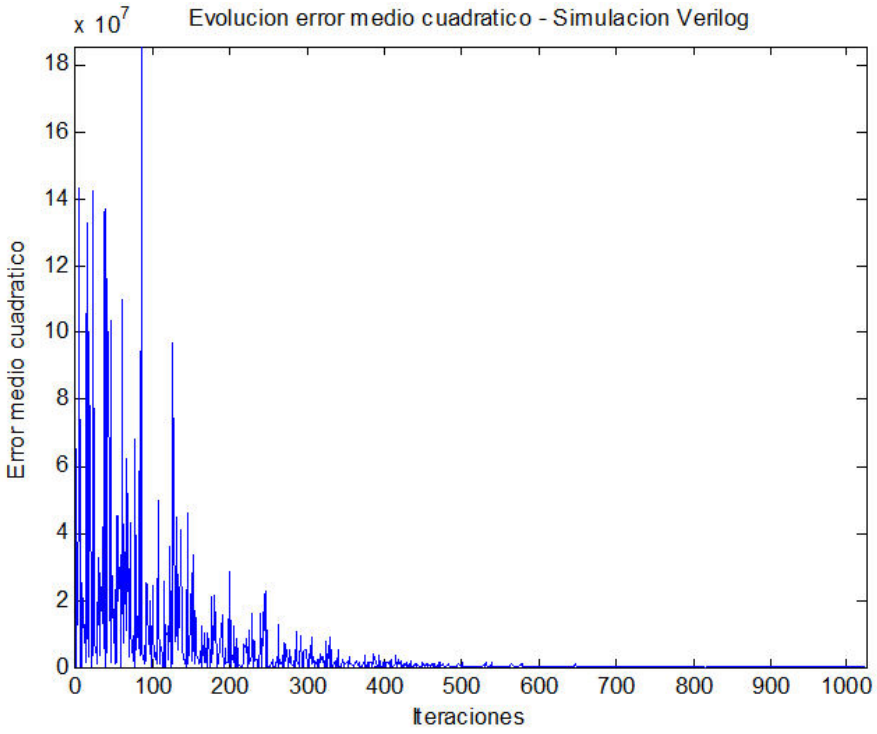


Fig. 5.10: Simulación error medio cuadrático

Se observa como el error tiende en una red neuronal desde valores elevados y un descenso de forma exponencial hasta llegar a pequeños valores, de aquí se puede concluir que con 1024 iteraciones, esta red compuesta por 5 sinapsis logra un estado de convergencia de sus pesos. En la Fig. 5.11, se muestra la evolución de los pesos participantes en esta red.

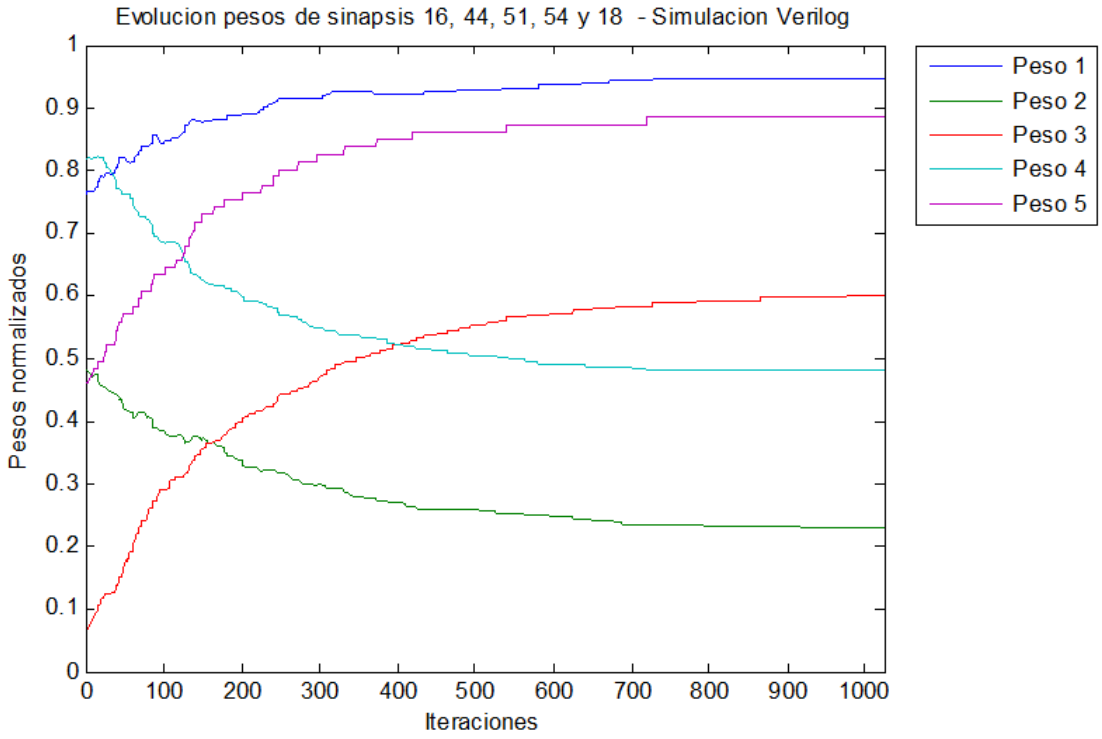


Fig. 5.11: Evolución pesos, red neuronal LMS

Se concluye mediante la observación de la evolución de los pesos de la Fig. 5.11, que se logra finalmente un estado de convergencia de la red bajo entrenamiento; esto sin embargo, no da luces que la implementación tenga validez en el desarrollo bajo cierta tecnología, es por esto que en siguiente capítulo se muestran resultados reales para validar definitivamente el funcionamiento del emulador.

5.6. Conclusiones

En este capítulo se han mostrado los resultados obtenidos en la simulación de varios módulos constituyentes de una red neuronal, las simulaciones son respaldadas por la teoría para confirmar que los bloques desarrollan sus funciones de manera apropiada.

La función tanh implementado en Verilog HDL, si bien produce buenos resultados en estas simulaciones; puede sin embargo, ser deficiente para otra implementación que requiera una aproximación con mayor precisión, para solucionar este problema, se puede recurrir a la alternativa de incrementar la cantidad de parámetros almacenados, pasando de este ejemplo del uso de una

RAM en bloque a pasar a 2 o más, lo que consumirá mayores recursos en la FPGA. La otra alternativa es la realización de aproximaciones de otro tipo, como aumento de orden de la ecuación de la recta, splines, etc.

Uno de los inconvenientes del multiplicador, radica en la latencia del cálculo cuando el peso ingresado como multiplicando corresponde a un índice muy alto, esto se debe a que la búsqueda se inicia desde el menor valor y se sube hasta llegar al nivel final, si es que el peso ingresado tiene esa propiedad. Una alternativa para mejorar esta situación, corresponde en una búsqueda simultánea desde los pesos de menor valor y desde los pesos de mayor valor, con esta solución la latencia del módulo se traspasa hacia los pesos de valor intermedio. Una segunda solución más radical corresponde al ingreso del peso como índice de los parámetros a, b y c; para esto, se tendría que modificar también la estructura del módulo de memoria.

Capítulo 6. Resultados en FPGA

6.1. Introducción

Las simulaciones mostradas en el capítulo anterior no reflejan necesariamente que la implementación en la FPGA sea satisfactoria, esto por los precarios resultados que pueden indicar un programa de simulación que no incluye observaciones con respecto a las características del hardware a utilizar, como son los retardos en las compuertas, latencia de multiplicadores hardware, lectura desde bloques de memoria, clock skew que se puede presentar en proyectos de gran envergadura, velocidad del reloj, tiempos de set-up, etc.

En el desarrollo de este trabajo se ocupó un FPGA de características particulares, se trata de la Virtex 2 Pro (Fig. 6.1), el cuál posee ciertas propiedades que se resumen a continuación.

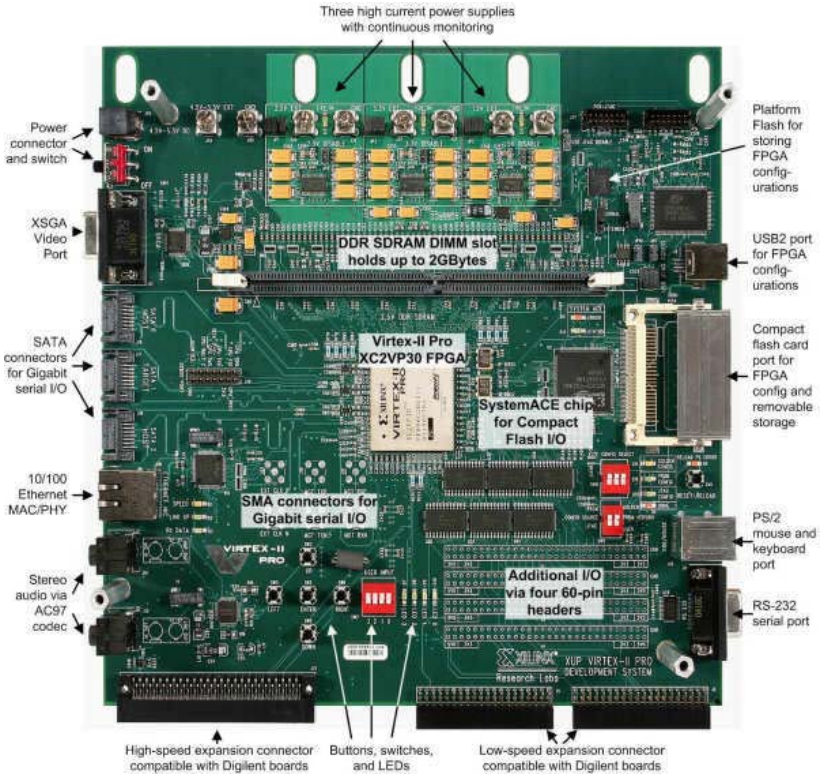


Fig. 6.1: Tarjeta Virtex 2 Pro

Algunas de las características más importantes se muestran en la tabla Tabla 6.1.

Celdas Lógicas	30816
Multiplicadores	136 de 18 bits
RAM en bloque	1024 x 18 bits por cada bloque
Procesadores	2 PowerPC
Socket RAM	1 DIMM DDR SDRAM hasta 2Gbytes
Puertos	Ethernet 10/100, USB2, Video XSGA, Audio, SATA, PS2 y RS232
Configuración	USB2, Compact Flash
Conectores de Expansión	de Alta y Baja velocidad
DCM	8

Tabla 6.1: Características principales Virtex 2 Pro

La frecuencia de operación del sistema es de 100MHz. A continuación se presenta el setup llevado a cabo en la mayoría de las pruebas realizadas en este capítulo.

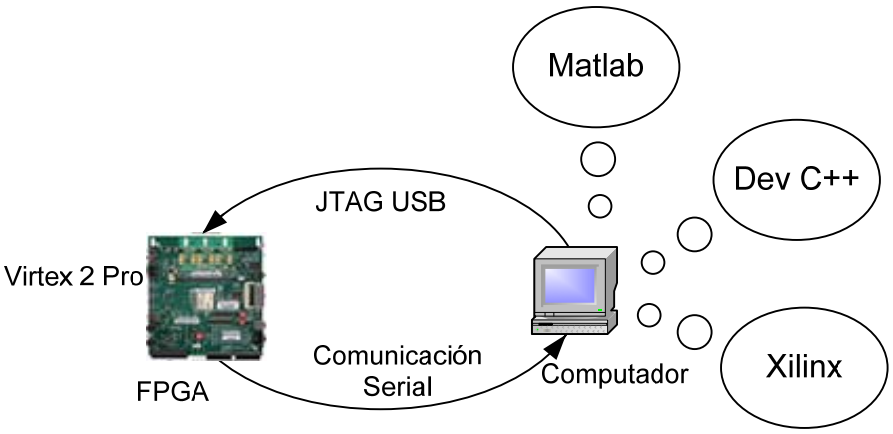


Fig. 6.2: Setup para pruebas

Con esta disposición, se carga la configuración desde la implementación en Verilog HDL hacia la tarjeta, posteriormente se envían los resultados de vuelta al computador mediante el enlace serial, en donde se reciben mediante la interfaz de comunicación serial programada en Dev C++, para esto se requiere un cable serial con esas características, posteriormente se despliegan los

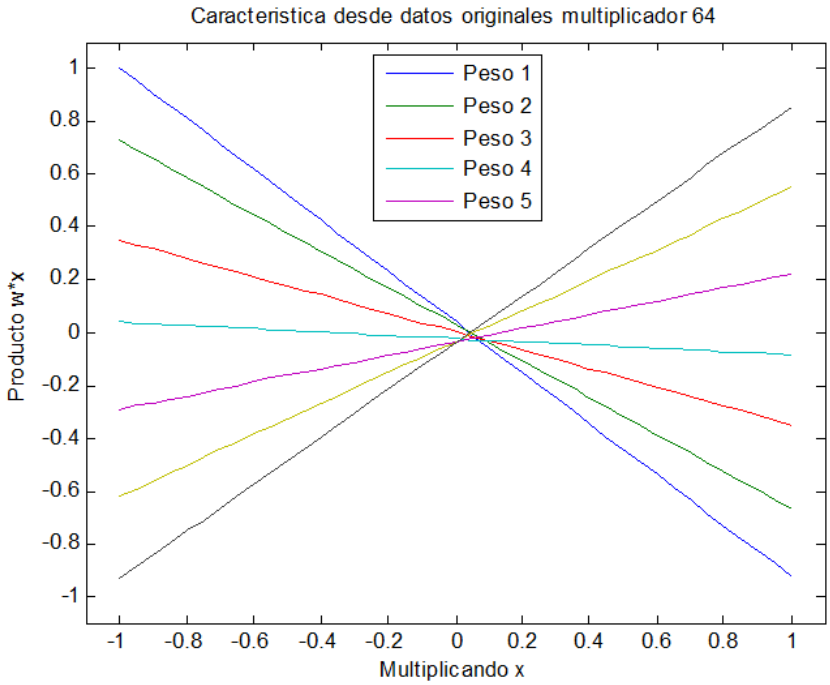
resultados obtenidos mediante script de Matlab® (Anexos A.1.i y A.1.k), encargado de abrir el archivo y según la organización de esta, recolectar los datos y graficarlos.

En las siguientes secciones se presentarán los resultados de la implementación del emulador, partiendo por los multiplicadores en la sección 6.2, sinapsis en sección 6.3 y red neuronal en sección 6.4. Se realiza un análisis de los recursos, velocidad y consumo de energía en la sección 6.5. Finalmente se entregan conclusiones de estos ensayos en la sección 6.6.

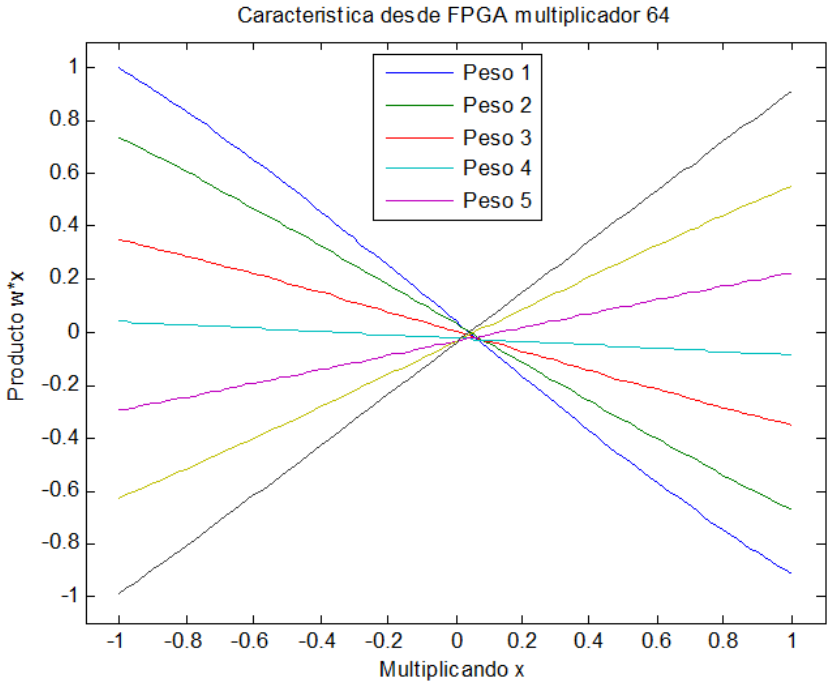
6.2. Multiplicadores

Siguiendo los conceptos de los párrafos finales de la sección anterior, se arma la configuración y utilizando el enlace serial se envían los datos de distintos multiplicadores.

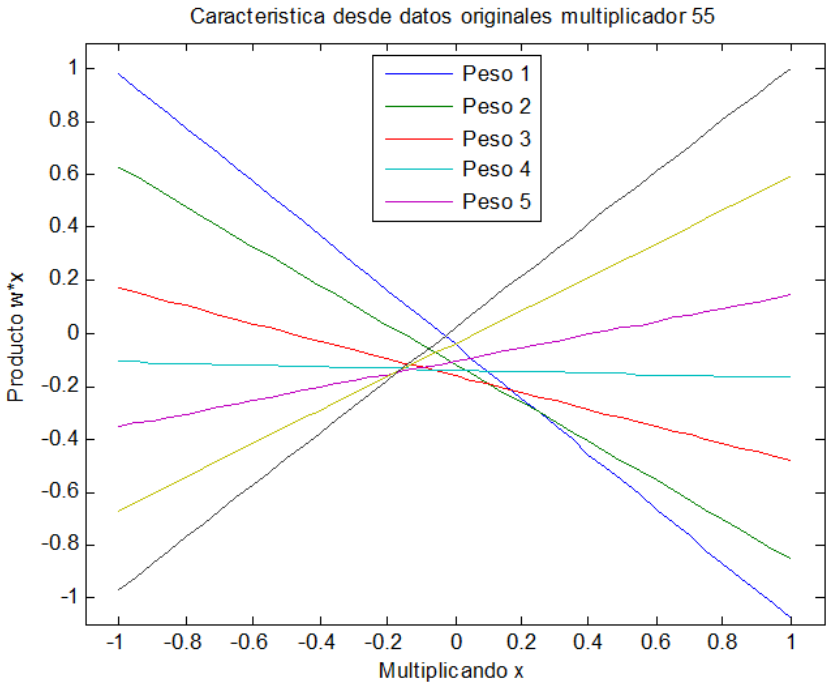
En la Fig. 6.3 se pueden comparar, los datos originales de mediciones de laboratorio, con el producto final de la emulación directamente en la FPGA.


Caracteristica desde datos originales multiplicador 64
Peso 1
Peso 2
Peso 3
Peso 4
Peso 5
Producto w*x
Multiplicando x
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

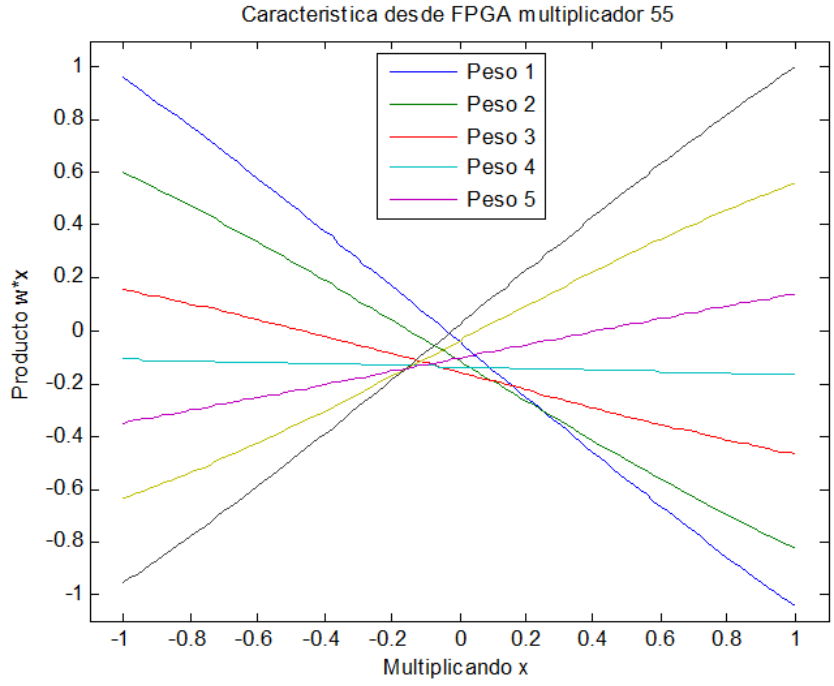
(a)


Caracteristica desde FPGA multiplicador 64
Peso 1
Peso 2
Peso 3
Peso 4
Peso 5
Producto w*x
Multiplicando x
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

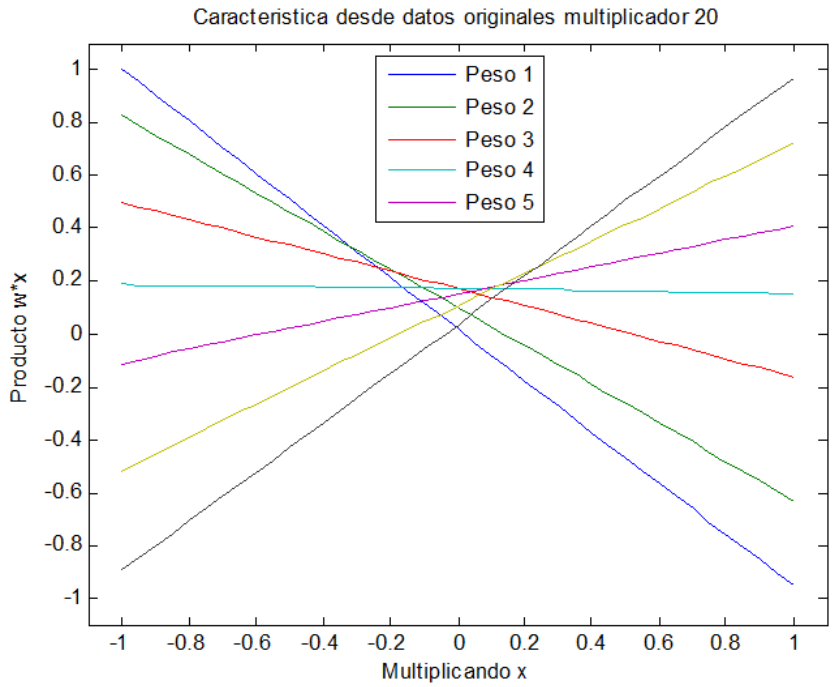
(b)


Caracteristica desde datos originales multiplicador 55
Peso 1
Peso 2
Peso 3
Peso 4
Peso 5
Producto w*x
Multiplicando x
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

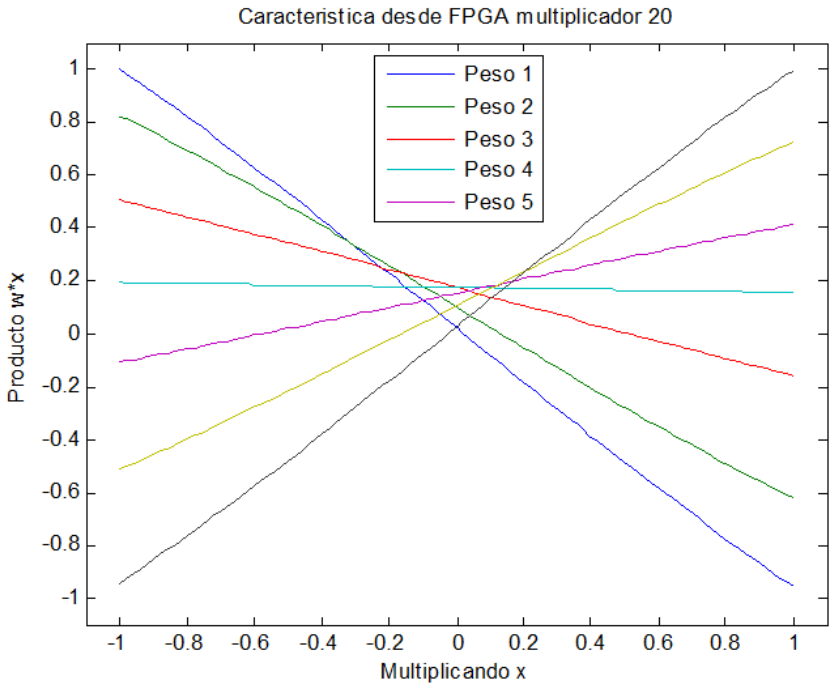
(c)


Caracteristica desde FPGA multiplicador 55
Peso 1
Peso 2
Peso 3
Peso 4
Peso 5
Producto w*x
Multiplicando x
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
-1 -0.8 -0.6 -0.4 -0.2 0 0.2 0.4 0.6 0.8 1

(d)



(e)



(f)

Fig. 6.3: Ensayos emulación multiplicador análogo, contraste con datos de laboratorio

(**a**) y (**b**), multiplicador 64; (**c**) y (**d**), multiplicador 55; (**e**) y (**f**), multiplicador 20

Para los gráficos previstos, se puede comprobar las grandes equivalencias entregadas por el diseño del emulador, hay que recalcar que las unidades del los datos de laboratorio son diferentes a los datos del emulador, pero los gráficos normalizados son los que determinan cuando una unidad tiene un comportamiento paralelo a otro.

6.3. Sinapsis

Con el fin de reunir los resultados de la sinapsis funcionando como tal y la celda de memoria, se realizan ensayos sobre la sinapsis para reducción de pruebas innecesarias, en la Fig. 6.4 se muestra el setup a realizar para probar mediante pantalla LCD los valores que retornan los módulos involucrados en esta implementación.

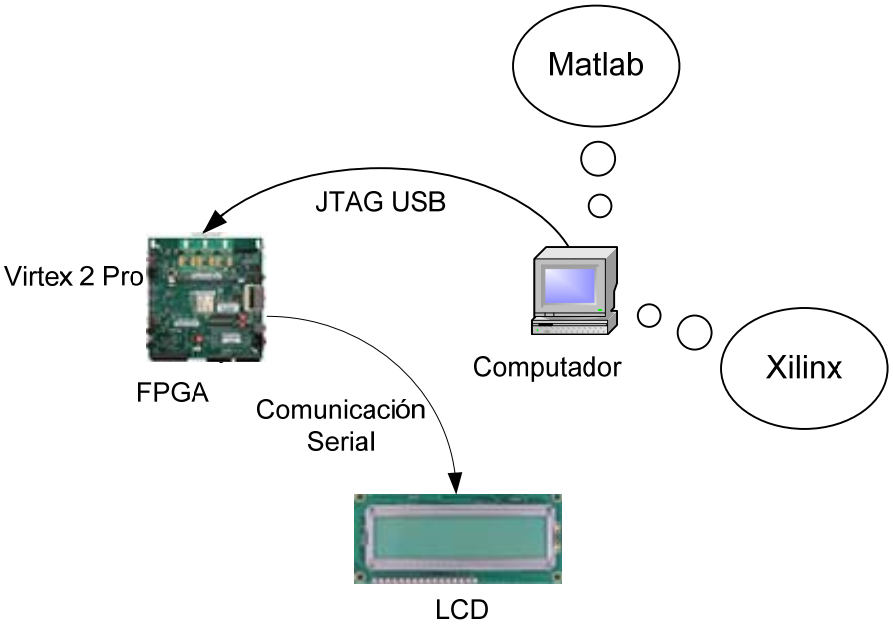
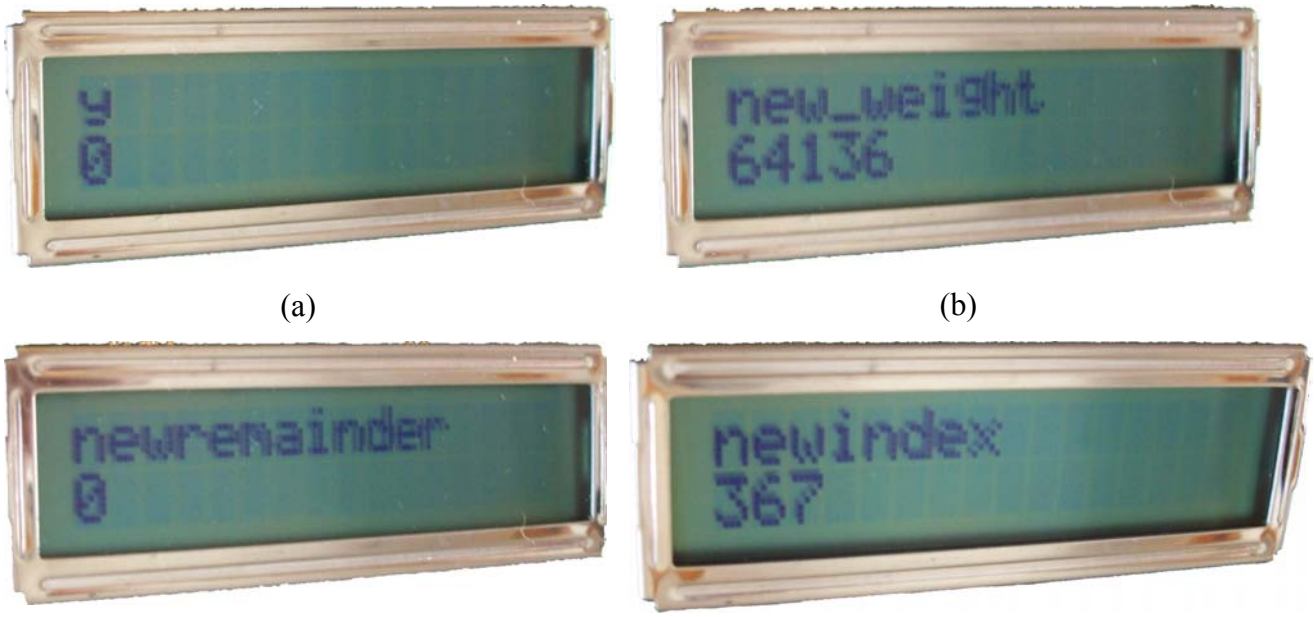


Fig. 6.4: Setup para pruebas de emulación de sinapsis

Las pruebas a desarrollar, involucran tanto el módulo de memoria, como el emulador del multiplicador análogo, las entradas para realizar el ensayo son las mismas a las vistas en la simulación de 5.4. A continuación se muestran algunas imágenes tomadas a la pantalla LCD recibiendo información de las variables del proceso en la FPGA siguiendo el esquema antes visto.

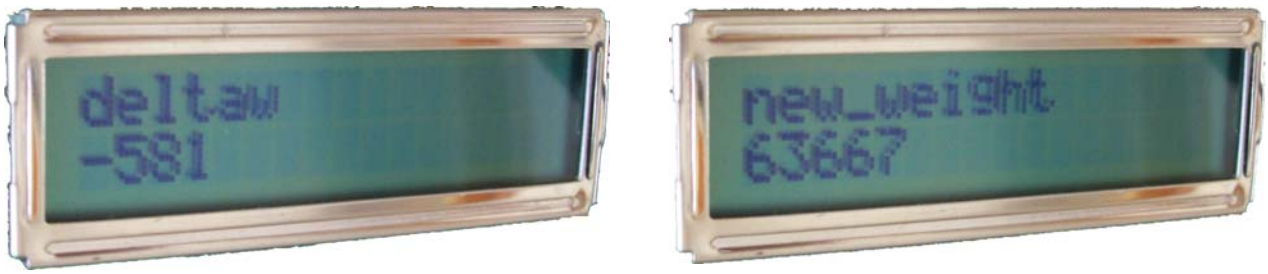


(a) (b)

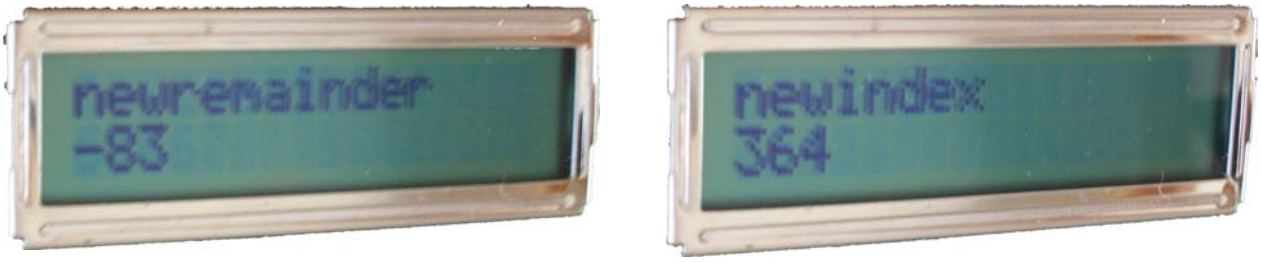
(c) (d)

Fig. 6.5: Emulación sinapsis – primera etapa, valores iniciales

En la Fig. 6.5, se puede notar como en las variables del módulo de memoria toman la condición inicial, donde la asignación del índice de memoria queda en 367, de forma análoga al capítulo de simulaciones, el peso correspondiente es de 64136 (FA88 en hexadecimal, revisar Tabla 5.1). En la Fig. 6.6, se presentan los resultados cuando se aplica una actualización de -581.



(a) (b)



(c) (d)

Fig. 6.6 Emulación sinapsis – segunda etapa, aplicación de Δw = -581

La Fig. 6.6 (a) demuestra la inyección de la actualización de -581, correspondiente a 3 pulsos y media carga, se demuestra mediante el resto de las imágenes de la pantalla LCD (Fig. 6.6 b, c y d), como se aplica de manera correspondiente el reajuste, en donde el remanente contiene la mitad de carga para generar un pulso y el peso (63667 - F8B3 hexadecimal) mantiene la coherencia con respecto a su simulación. Continuando las pruebas de emulación; se presenta en la Fig. 6.7, el efecto de la memoria al ingresar un delta de peso de -747.

(a) (b)

(c) (d)

Fig. 6.7: Emulación sinapsis – tercera etapa, aplicación de Δw = -747

Se concluye finalmente que el módulo de memoria es verificado por mantener su funcionamiento congruente con la simulación de la sección 5.4, en este apartado se aplica una actualización correspondiente a 4 pulsos y medio, como el remanente anterior es equivalente a

medio pulso, este se suma al delta y genera un total de 5 pulsos, resultando en un remanente igual a cero, el índice también así lo indica, bajando desde 364 en su valor en la etapa anterior, hasta 359. El peso correspondiente a ese índice es de 62848 (F580 en hexadecimal). Cerrando definitivamente con la aprobación de este módulo para la emulación de una sinapsis. Se termina esta sección probando la funcionalidad de la sinapsis invocando una multiplicación en su módulo correspondiente.

(a) (b)

(c)

Fig. 6.8: Emulación sinapsis – cuarta etapa, multiplicación peso anterior con x = 20000

Los resultados para las entradas utilizadas igualmente en la sección 5.4, mostradas en la Fig. 6.8, corroboran que la multiplicación se lleva a cabo de forma satisfactoria, se puede entonces permitir la utilización del módulo de sinapsis para la configuración de una red adaptiva con las características que el usuario estime convenientes, se termina este capítulo presentando resultados ante la construcción definitiva de una red neuronal.

6.4. Red neuronal

Finalmente se muestran los resultados de la evolución de los pesos que se mostraron en la sección 5.5, esta vez con el emulador mismo funcionando en la Virtex 2 Pro, la disposición de dispositivos es la mostrada en la Fig. 6.2. En la Fig. 6.9 se muestra la evolución del error medio cuadrático.

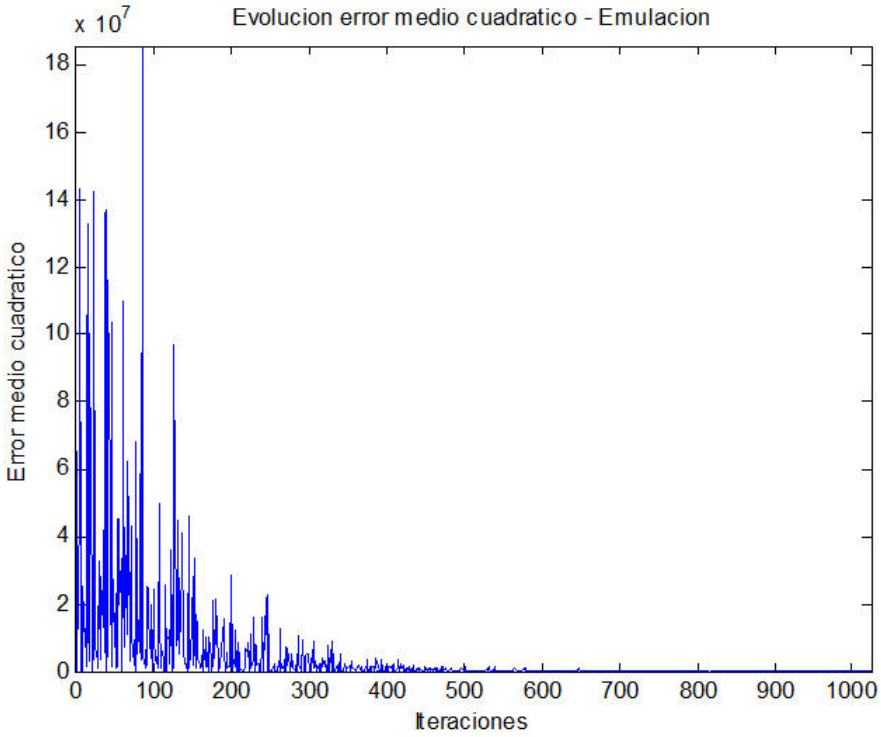


Fig. 6.9: Error medio cuadrático – Emulación.

De forma análoga, se observa que los resultados encontrados son altamente similares a los de la Fig. 5.10, por lo tanto de igual forma se concluye que el progreso de los pesos alcanza un estado de convergencia antes de la aplicación de los 1024 puntos, a continuación se muestra en la Fig. 6.10 los resultados de la emulación de una red adaptiva aplicando algoritmo LMS para actualización de pesos.

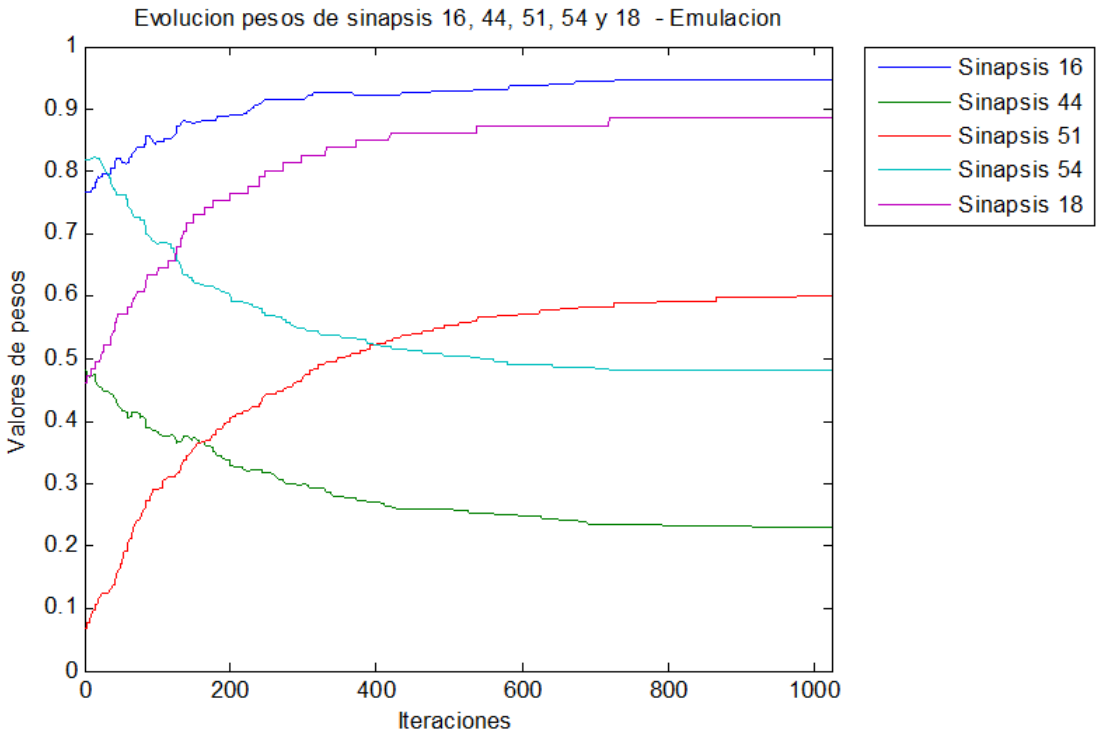


Fig. 6.10: Emulación de evolución de pesos en red adaptiva

Se comprueba definitivamente, la validación de la emulación de una red adaptiva utilizando los bloques principales de algoritmos de redes neuronales en VLSI.

En el apartado a continuación, se arrojan resultados de carácter de implementación en FPGA, que comprueban la conveniencia de realizar un emulador de circuitos adaptivos en silicio.

6.5. Sumario de diseño en FPGA

Al igual que los resultados de la emulación de la red adaptiva, los aspectos de diseño son muy importantes en la implementación de hardware funcional. Esto con el fin de determinar si el esquema realmente integra ventajas al funcionar de forma autónoma. En las siguientes secciones se abordan distintas consideraciones, como son los recursos utilizados en sección 6.5.1, las velocidades de operación en sección 6.5.2. Finalmente se evalúa el consumo de energía tanto en funcionamiento como en estado de reposo, en sección 6.5.3.

6.5.1. Consideraciones de Recursos

Las pruebas realizadas para 5 sinapsis en la red neuronal arrojan los siguientes resultados de utilización de recursos en la tarjeta.

HDL Synthesis Report		937x17-bit ROM	: 2	17-bit subtractor	: 10
		# Multipliers	**: 65**	18-bit adder	: 15
Macro Statistics		15x18-bit multiplier	: 10	18-bit addsub	: 5
# ROMs	**: 30**	17x10-bit multiplier	: 5	18-bit subtractor	: 5
1024x17-bit ROM	: 5	17x15-bit multiplier	: 10	34-bit adder	: 20
1025x17-bit ROM	: 2	17x17-bit multiplier	: 20	36-bit adder	: 2
1049x17-bit ROM	: 2	17x18-bit multiplier	: 10	36-bit subtractor	: 1
1469x17-bit ROM	: 2	18x12-bit multiplier	: 5	4-bit adder	: 1
236x17-bit ROM	: 1	22x11-bit multiplier	: 5	5-bit adder	: 10
258x17-bit ROM	: 1	**# Adders/Subtractors**	**: 143**	5-bit adder carry out	: 5
264x17-bit ROM	: 1	10-bit adder	: 11	7-bit subtractor	: 3
313x17-bit ROM	: 2	10-bit adder carry out	: 10	9-bit adder	: 15
369x17-bit ROM	: 1	11-bit addsub	: 10	**# Counters**	**: 1**
512x32-bit ROM	: 10	14-bit adder	: 5	9-bit up counter	: 1
80x17-bit ROM	: 1	17-bit adder	: 15	**# Registers**	**: 331**

1-bit register	: 62	8-bit register	: 2
10-bit register	: 26	9-bit register	: 7
11-bit register	: 10	**# Comparators**	**: 66**
13-bit register	: 10	11-bit comparator greater	: 20
14-bit register	: 5	11-bit comparator less	: 1
15-bit register	: 10	15-bit comparator greater	: 5
16-bit register	: 6	15-bit comparator less	: 5
17-bit register	: 75	16-bit comparator greater	: 5
18-bit register	: 46	16-bit comparator less	: 5
22-bit register	: 5	18-bit comparator greatequal	: 10
32-bit register	: 10	18-bit comparator less	: 10
34-bit register	: 40	6-bit comparator not equal	: 5
36-bit register	: 3	**# Priority Encoders**	**: 2**
4-bit register	: 1	36-bit 1-of-5 priority encoder	: 2
5-bit register	: 10	**# Xors**	**: 5**
7-bit register	: 3	1-bit xor2	: 5

Tabla 6.2: Utilización de recursos Virtex 2 Pro

Esta información además se muestra resumida en la Fig. 6.11.

Device Utilization Summary				
Logic Utilization	**Used**	**Available**	**Utilization**	**Note(s)**
Number of Slice Flip Flops	16	27,392	1%	
Number of 4 input LUTs	18	27,392	1%	
Logic Distribution				
Number of occupied Slices	16	13,696	1%	
Number of Slices containing only related logic	16	16	100%	
Number of Slices containing unrelated logic	0	16	0%	
Total Number of 4 input LUTs	26	27,392	1%	
Number used as logic	18			
Number used as a route-thru	8			
Number of bonded IOBs	3	556	1%	
Number of PPC405s	0	2	0%	
Number of GCLKs	1	16	6%	
Number of DCMs	1	8	12%	
Number of GTs	0	8	0%	
Number of GT10s	0	0	0%	
Total equivalent gate count for design	7,287			
Additional JTAG gate count for IOBs	144			

Fig. 6.11: Resumen utilización de recursos Virtex 2 Pro

Como se observa en el resumen general de la Fig. 6.11; la capacidad de la tarjeta utilizada no se ve sobrepasada en el diseño de 5 sinapsis, por lo tanto se tiene la libertad de incluir una red neuronal de dimensiones considerables. Lo que permite además la futura integración de más perceptrones y capas de niveles superiores.

Unas de las unidades de mayor relevancia en la síntesis en un FPGA corresponden a los multiplicadores de hardware y las memorias de sólo lectura (ROM). El resumen de la Tabla 6.2 indica que la cantidad de memorias ROMs utilizadas corresponden a 30; de estas memorias, 4 son utilizadas por cada sinapsis adaptiva (2 en multiplicador análogo, 1 en celda de memoria), 1 en memoria de vectores de entrada, y 2 por cada multiplicador de pesos fijos. Entregando un total de 6 memorias ROMs por sinapsis. En esta experiencia, se utilizaron 5 sinapsis, por lo tanto, el total asciende a 30 unidades de memoria ROM.

Con respecto a los multiplicadores de hardware, son utilizados 6 por cada sinapsis adaptiva (5 en multiplicador análogo, 1 en celda de memoria), 5 en cada multiplicador análogo de peso fijo, y 2 multiplicadores de hardware en cálculo de incremento de peso, dando un total de 13 multiplicadores por sinapsis, considerando que son declaradas 5 sinapsis, el total asciende a 65 multiplicadores de hardware.

Hay que recordar que los recursos tabulados, corresponden al emulador con la generación de referencia integrada para observar los resultados, es decir, existe por cada sinapsis adaptivo, un multiplicador de peso fijo, lo que introduce Flip-Flops, compuertas lógicas, multiplicadores de hardware y celdas de memoria descritas como en el párrafo anterior. Estos no se utilizan en una prueba real del emulador, por lo tanto se puede deducir que los recursos en uso son aun menores, consiguiéndose mayores capacidades para generar redes más sofisticadas.

En la siguiente sección, se detallan los resultados en velocidad que se pueden lograr con la tecnología disponible.

6.5.2. Consideraciones de Velocidad

El software de implementación en FPGA Xilinx® ISE™ Webpack™ 9.2.04i indica además que el camino crítico detectado, induce una limitante en la frecuencia de reloj a utilizar, la cual no debe poseer esta un periodo más rápido que 3,028ns. Por lo tanto, se puede arrancar el emulador a una velocidad máxima de reloj de 330,24MHz.

Lamentablemente la velocidad de reloj sistema de la Virtex 2 Pro es de 100MHz, por lo tanto el periodo de trabajo es de 10ns. La simulación de la red de 5 sinapsis indica que la cantidad de pulsos utilizada para llegar a un estado de estabilidad con 1024 iteraciones es de 542229 pulsaciones, esto combinado con el máximo periodo de reloj del sistema (10ns), muestra que el tiempo de convergencia de una red de 5 sinapsis y un perceptrón es de 5,4m[s].

Si es conectado al FPGA un oscilador con las características de la frecuencia máxima permisible (330,24MHz); considerando la cantidad de pulsos necesaria para una buena convergencia, se estima un tiempo de estabilización de 1,6m[s], siendo este tiempo mucho menor al logrado utilizando solamente los recursos de la Virtex 2 Pro.

Se define a continuación, la carga de poder que requiere la FPGA para funcionar de forma adecuada, bajo las características antes mencionadas.

6.5.3. Consideraciones de Energía

El análisis de consumo de energía se lleva a cabo gracias al accesorio XPower 9.2.04i, propiedad de Xilinx™, el informe indica que el consumo para una frecuencia de reloj de 100MHz y la activación de aprendizaje de la red neuronal cada 1kHz es de 112mW. Cuando el emulador se encuentra en estado stand-by, la FPGA muestra un consumo aproximado de 103mW.

6.6. Comentarios y Conclusiones

A partir de los resultados obtenidos de las experiencias vividas en la realización de estos ejemplos, es posible entregar algunos comentarios y conclusiones al respecto.

A pesar que la emulación hardware podría presentar diferencias con respecto a las simulaciones del capítulo anterior, en esta sección se demuestra que la herramienta construida provee la misma funcionalidad que lo predicho en teoría. Los multiplicadores por otro lado, demuestran pequeñas diferencias con respecto a los datos de laboratorio. Esto se justifica por la aproximación entregada por las constantes *a*, *b* y *c*. Lo que realmente interesa de la emulación del multiplicador, es la analogía lograda con la tanh de Matlab® utilizando los parámetros que aproximan los datos originales de laboratorio. En ese aspecto, los multiplicadores emulados muestran 0 error.

La evolución de los pesos muestra características poco suaves en las curvas observadas tanto en la simulación de la Fig. 5.11, como en la emulación de la Fig. 6.10. Esto se explica porque las celdas de memoria almacenan ciertos valores de pesos; la distancia entre un valor de peso y otro, depende de la ganancia que la memoria contiene al aplicar un pulso, es por esto que las celdas que contienen altos niveles de valores de pesos (baja ganancia al aplicar un pulso), pueden indicar tendencias, en los algoritmos adaptivos, de formas suaves. En cambio, las memorias que no contienen una vasta cantidad niveles (alta ganancia al aplicar un pulso), muestran pobres curvas; esto sin embargo, no afecta de gran forma el funcionamiento del algoritmo, debido a la acción de los remanentes de carga existentes en cada celda de memoria. La ausencia de los remanentes de carga provocaría el deterioro del funcionamiento, de forma que la evolución sería más lenta, teniéndose que elevar la tasa de aprendizaje y con las consecuencias de producir mayor inestabilidad o aumentar el MSE por los asuntos de desajuste especificados en el capítulo 2.

La emulación de un algoritmo que entrena los pesos en un perceptrón ha sido completamente tratada. En el siguiente capítulo se entregan las conclusiones finales de este trabajo, dando además los posibles caminos a seguir en el tema de la emulación de algoritmos de aprendizaje.

Capítulo 7. Conclusiones

En este capítulo se enuncian las conclusiones obtenidas del trabajo y se analizan las posibles direcciones de futuros trabajos sobre el tema.

7.1. Sumario

Los alcances del proyecto fueron totalmente cubiertos. Entre los más relevantes, se puede mencionar que:

- Se implementó una serie de funciones en Verilog HDL que permiten integrar los modelos ideales de los algoritmos con las unidades funcionales utilizadas en VLSI, con esto se puede evaluar el comportamiento del algoritmo y desarrollar técnicas de compensación.
- Se desarrolló un script en ambiente Matlab® que realiza, en términos generales, las funciones de: generar los parámetros de las unidades aritméticas con las correspondientes memorias de almacenamiento de funciones; comandar la implementación en Verilog HDL de las estructuras que el usuario desea implementar; y recibir resultados tanto de simulación en Xilinx® ISE™ Webpack™, como de la emulación enviada a través del enlace serial.
- Se programó una interfaz de comunicación serial basada en Windows API C para tomar los resultados de la FPGA, y entregarlos en archivo para poder ser procesados mediante el script de Matlab®.

Se debe considerar que la limitación de los resultados obtenidos, son en base a la aplicación de vectores del universo de entrada provenientes de valores aleatorios generados en Matlab®, y no desde una interacción directa con alguna circuitería adaptiva o fenómeno natural. Queda entonces, la interrogante de conocer el comportamiento de un esquema externo usando un algoritmo adaptivo por medio del emulador aquí construido.

La tasa de aprendizaje es fijada durante todo el proceso de aprendizaje, esto limita la velocidad y convergencia en la red neuronal. Sin embargo, el funcionamiento del algoritmo puede

ser igualmente evaluado, posibilitando al usuario conseguir mejores resultados si este ajusta esta tasa y/o la hace variar en el tiempo (reduciéndola).

A pesar de los puntos anteriores, se entrega al usuario final, una alta libertar de experimentar distintas configuraciones que permitan apreciar cuando un algoritmo; mediante su emulación, puede o no ser realizado en VLSI análogo.

7.2. Conclusiones

Gracias a los estudios de desarrollo e implementación llevados a cabo en este trabajo, se puede vislumbrar que la emulación desplegada en los capítulos anteriormente vistos, abre un nuevo horizonte hacia el acercamiento de implementaciones de redes adaptivas en VLSI. Las conclusiones derivadas de este trabajo se enuncian como sigue.

Los recursos necesarios para poder llevar la herramienta de decisión (emulador) a un nivel superior se prevé completamente viable, esto gracias a los bajos consumos observados en el resumen de la Fig. 6.11, ahí se puede notar que la mayoría de los conceptos poseen utilización de orden de 1%, dando luces a la posibilidad de emular complejas estructuras de redes. En donde es posible adicionar etapas de pre-procesamiento, como por ejemplo redes de decorrelación para acelerar la convergencia de los pesos; aumentar la cantidad de perceptrones e incluso adicionar niveles de capas para realizar selecciones más abstractas. Pudiéndose implementar otras técnicas de aprendizaje como back-propagation.

La velocidad con la cual la red adaptiva logra su estado de convergencia es de unos pocos mili segundos (5,6ms en el caso de 5 sinapsis, sección 6.5.2). Pero sin embargo, esto depende fuertemente de la naturaleza del fenómeno a clasificar, y de la tasa de aprendizaje. Se concluye de esto que la gran velocidad obtenida permite emular circuitos adaptivos importantes en lo que respecta a aplicaciones como reconocimiento de rostros [15], voces humanas [16], o escritura manuscrita, con sorprendentes resultados.

Aunque el emulador construido soporta una velocidad de reloj superior a la ofrecida en la Virtex 2 Pro, la configuración del emulador sin oscilador es suficiente para mencionar que no se dispone de un marco de comparación con una simulación desarrollada en Matlab®. Esto responde a la medición del tiempo de CPU que toma el simulador en entrenar un perceptrón con las mismas

consideraciones empleadas en el emulador, correspondientes a cantidad de sinapsis y muestras del universo de entrada. Medición que indica que el algoritmo requiere 2,7908s en terminar el periodo de entrenamiento.

Es sabido que Matlab® tarda mucho tiempo en realizar las operaciones por ser este un lenguaje intérprete, es por esto que se decide hacer la medición de tiempo de CPU que tomaría equivalentemente un simulador escrito en C (lenguaje compilador), realizando una cantidad de operaciones que realiza similarmente el simulador. Obteniéndose un tiempo de 15,5116ms para las mismas condiciones anteriores. La eficiencia en tiempo lograda con respecto a diferentes pruebas software y hardware (FPGA) se resume en la Tabla 7.1.

Se puede entonces reafirmar que la solución mediante FPGA además de ofrecer las ventajas propias antes comentadas por la emulación, entrega resultados en tiempos menores a las soluciones software, a pesar que este aspecto no es de principal interés en el desarrollo de este trabajo.

	Matlab	Equivalente C	FPGA 100Mhz	FPGA 330,24MHz
Tiempo (ms)	2790,8	15,5	5,6	1,6
Eficiencia				
Matlab	0,00%	-99,44%	-99,80%	-99,94%
Equivalente C	17905,16%	0,00%	-63,87%	-89,68%
FPGA 100MHz	49735,71%	176,79%	0,00%	-71,43%
FPGA 330,24MHz	**174325,00%**	**868,75%**	**250,00%**	0,00%

Tabla 7.1: Rendimientos en términos de tiempo de ejecución

Es requerimiento de las pruebas software que se destaque además las características del computador con los cuáles se obtuvieron los resultados de la Tabla 7.1, ya que los tiempos pueden ser más tentadores o menos ostentosos si el computador utilizado para obtener los resultados difiere de los considerados clave en la Tabla 7.2.

Sistema Operativo	Microsoft Windows Vista Ultimate
Tipo de Procesador	Mobile DualCore Intel Core 2 Duo T7200, 2000 MHz / 4MB Cache L2
Chipset placa Base	Mobile Intel Calistoga-PM i945PM
Memoria del Sistema	2048 MB (DDR2-667 DDR2 SDRAM)
Versión de Matlab	7.4.0.287 (R2007A)

Tabla 7.2: Características computador

Tanto en el capítulo de simulación (5), como en el de emulación (6), la adaptación de los pesos mostró la tendencia a la convergencia, el error medio cuadrático tiende a 0 en ambas situaciones, este valor es ideal ya que la emulación no supone la introducción de ruido Gaussiano como en el esquema de la Fig. 2.4. Se concluye de esto, que el MSE en la práctica tendrá valores reducidos pero jamás se obtendrá la perfección de error cero.

Si se comparan las implementaciones en software y hardware, se puede comentar que la evaluación desarrollada en el simulador (software), permite decidir con cierto grado de seguridad, la posibilidad que un proceso en VLSI pueda tener éxito en su funcionamiento; en cambio, con el emulador (hardware) desarrollado en esta memoria de título, se tendrá la plenitud de decidir si el esquema bajo prueba funcionará, todo esto gracias a la clase de ensayos a la cual puede estar sometido un emulador.

Se ha demostrado la efectividad de un sistema de VLSI análogo, pero se debe recordar, que la emulación es realizada en un sistema completamente digital, como lo es un FPGA. Cabe entonces señalar, que ambos espectros son complementados para poder efectuar un estudio de implementación de un circuito adaptivo, se concluye de esto, que el emulador es en algún nivel, de señal mixta.

Las pruebas experimentales de la herramienta hardware mostraron excelentes resultados (reducido MSE y pesos convergentes), que permitieron observar los efectos del hardware en el desempeño de los algoritmos. Posibilitando la evaluación y desarrollo de sistemas de procesamiento adaptivo de mediana y gran escala antes de su diseño y fabricación final, de una forma sencilla y práctica.

7.3. Trabajo Futuro

La continuación de este trabajo, está dirigida principalmente al acoplamiento de herramientas al emulador, como pueden ser: la adición de una interfaz de operación; donde se puedan escoger los algoritmos a desarrollar, como por ejemplo PCA, ICA, SOM, etc. e incorporando, si es necesario, nuevas técnicas de compensación y unidades aritméticas o de procesamiento adicionales a las ya utilizadas. La elección de las sinapsis puede incluir gráficas de la celda de memoria y barrido representativo del multiplicador, con el fin que el usuario tenga plena conciencia del tipo de hardware a emular. Otra muy buena alternativa consiste en realizar la adaptación a trabajar con datos de laboratorio diferentes a los dispuestos.

Bibliografía

[1] Chris Diorio, David Hsu, and Miguel Figueroa. "Adaptive cmos: from biological inspiration to systems-on-a-chip," in *Proceedings of the IEEE,* 2002, vol 90, pp. 345-357.

[2] Miguel Figueroa, Seth Bridges, and Chris Diorio. "On-chip compensation of device-mismatch effects in analog vlsi neural networks," in *Advances in Neural Information Processing Systems 17*, Lawrence K. Saul, Yair Weiss, and Léon Bottou, Eds. MIT Press, Cambridge, MA, 2005.

[3] Esteban J. Matamala. "Simulación de Algoritmos de Procesamiento Adaptivo de Señales en VLSI", Memoria de Título, Ingeniero Civil Electrónico, Marzo 2006, Departamento de Ingeniería Eléctrica, Facultad de Ingeniería, Universidad de Concepción.

[4] Chris Duffy and Paul Hasler, "Scaling pFET Hot-Electron Injection.," 2004.

[5] Figueroa M,. Bridges S., Hsu D., and Diorio C., "A 19.2GOPS Mixed-Signal Filter with Floating-Gate Adaptation," vol. 39, pp 1196-1201, 2004.

[6] Abedin, K. Kamimura, A. Ahmadi, H. J. Mattausch and T. Koide., "Fully-Parallel Associative Memory Architecture Realizing Minimun Euclidean Distance Search.,".

[7] K. N. Salama, A. M. El-Tawil, A. M. Soliman and H. O. Elwan., "CMOS Programmable Imager Implementing Pre-Processing Operations.," vol 19, pp. 279-293, 1999.

[8] Gilbert B., "A Precise four-quadrant multiplier with sub-nanoseconds response," pp. 3-365, 1968.

[9] Bernard Widrow and Michael Lehr., "30 years of adaptive neural networks: Perceptron, madaline and backpropagation," in Proceedings of the IEEE, 1990, vol. 78.

[10] Larry L. Horowitz and Kenneth D. Senne, "Performance advantage of complex lms for controlling narrow-band adaptive arrays," in IEEE Transactions on Circuits and Systems, 1981, vol. CAS-28, pp. 562–576.

[11] Haykin S., *Neural Networks, a Comprehensive Foundation.*, Prentice Hall, 2nd edition, 1999.

[12] Kilts S., *Advanced FPGA Design, Architecture, Implementation, and Optimization.*, Wiley-Interscience, 2007.

[13] Xilinx®, *Xilinx University Program Virtex-II Pro Development System.*, Hardware Reference Manual, 2005.

[14] P. Churchland and T. Sejnowski, *The Computational Brain*. Cambridge, MA: MIT Press, 1993.

[15] G. Carvajal, W. Valenzuela and M. Figueroa, "Subspace-Based Face Recognition in Analog VLSI", Advances in Neural Information Processing Systems 20. Cambridge, MA: MIT Press, 2008.

[16] Antonio Pedro Timoszczuk and Euvaldo F. Cabral Jr., "Speaker Recognition Using Pulse Coupled Neural Networks", in *Proceedings of International Joint Conference on Neural Networks*, pp 1965–1969, 2007.

Anexo A. Código Fuente

A.1 Scripts Matlab

En este apartado, se detalla el código fuente de los scripts que generan los pesos y parámetros a b c, para su posterior uso en FPGA, se construyen los parámetros de la ecuación de la recta y se guardan para su acceso posterior desde módulo de memoria. Se contrastan los resultados de mediciones de laboratorio, tanh de Matlab® y tanh por rectas.

Se genera el código Verilog HDL que especifica los tamaños requeridos en los módulos de memoria para almacenar el valor de los pesos para el caso de las celdas de memoria y los parámetros a b c para el caso de los multiplicadores. Se genera además el código para configurar una red neuronal con consideraciones ingresadas por consola. Finalmente se crean los vectores del universo de entrada para excitar los pesos fijos que entrenarán el perceptrón adaptivo.

A.1.a Extracción de pesos y constantes abc

```
% Cierra/borra todo y limpia linea de comando
close all;
clc;
clear all;

delete C:\EmuVerilog\labdata.txt
delete C:\EmuVerilog\memorydata.txt

load array_data.mat
load Base_Datos_multiplicador_A_tanh(B_X)+C.mat
load array_data.mat

FID2 = fopen('C:\EmuVerilog\labdata.txt','w');
FID4 = fopen('C:\EmuVerilog\memorydata.txt','w');

if (FID2 == -1)
   error('No se pudo abrir ni crear archivo labdata');
end
if (FID4 == -1)
   error('No se pudo abrir ni crear archivo memorydata');
end

count = fprintf(FID2,'\n@000\r'); %Inicio de datos en memoria
count = fprintf(FID4,'\n@000\r'); %Inicio de datos en memoria
[rows,columns] = size(Datos_W);

FID_mult = zeros(columns,1);
```

```
FID_mems = zeros(columns,1);

for i = 1 : columns, %Creacion de archivos con datos de memorias y multiplicadores
   delete(sprintf('C:\\EmuVerilog\\data_mult%i.txt',i));
   FID_mult(i) = fopen(sprintf('C:\\EmuVerilog\\data_mult%i.txt',i),'w');

   delete(sprintf('C:\\EmuVerilog\\data_mem%i.txt',i));
   FID_mems(i) = fopen(sprintf('C:\\EmuVerilog\\data_mem%i.txt',i),'w');
end

%Se obtienen los maximos y minimos de los parametros a, b y c para
%transformacion a datos en verilog
maximoa = max(max(Datos_C(:,1,:))); maximob = max(max(Datos_C(:,2,:))); maximoc = max(max(Datos_C(:,3,:)));

minimoa = min(min(Datos_C(:,1,:))); minimob = min(min(Datos_C(:,2,:))); minimoc = min(min(Datos_C(:,3,:)));

minabc = min([minimoa minimob minimoc]); maxabc = max([maximoa maximob maximoc]);
deltaabc = maxabc - minabc;
nbitsabc = 16+1;                    %16 bits datos mas bit signo (17 bits)
stepabc = deltaabc / (2^(nbitsabc-1)-1);

nbitsb = 14 + 1;                    %14 bits para representar numero mas un bit de signo
delta_b = maximob;                  %Se considera valor maximo de b, delta desde 0 a maxima magnitud
step_b = delta_b / (2^(nbitsb-1)-1); %Step se calcula sin bit de signo

max_value=max(max(Datos_W(2:rows,:)));
min_value=min(min(Datos_W(2:rows,:)));
delta = max_value - min_value;

nbits = 16;
step = delta / (2^(nbits)-1);
existe_redundancia2 = 0;
normalized_num_k1 = (2^(nbits)-1)/2;
deltas2_reg = zeros(columns,1);

for i = 1 : columns, %almacenamiento de parametros y pesos por cada multiplicador/memoria
   count = fprintf(FID2,'\n%s\r',dec2hex(i));    count = fprintf(FID4,'\n%s\r',dec2hex(i));
   sweeps = Datos_W(1,i);
   count = fprintf(FID2,'\n%s\r',dec2hex(sweeps));    count = fprintf(FID4,'\n%s\r',dec2hex(sweeps));

   count = fprintf(FID_mult(i),'\n@000\r');
   count = fprintf(FID_mult(i),'\n%s\r',dec2hex(sweeps));

   count = fprintf(FID_mems(i),'\n@000\r');
   count = fprintf(FID_mems(i),'\n%s\r',dec2hex(sweeps));
   for j = 2 : sweeps+1,
      norma = round(Datos_C(j-1,1,i)/stepabc);
      normb = round(Datos_C(j-1,2,i)/step_b);
      normc = round(Datos_C(j-1,3,i)/stepabc);
      if (norma < 0),
         numbera = bitcmp(abs(norma),nbitsabc) + 1;  %Si el numero es negativo se obtiene complemento a 2
      else
         numbera = norma;    %Si el numero es positivo se deja como tal
      end
      if (normb < 0),
         numberb = bitcmp(abs(normb),nbitsb) + 1;    %Si el numero es negativo se obtiene complemento a 2
      else
```

```
        numberb = normb;    %Si el numero es positivo se deja como tal
      end
      if (normc < 0),
        numberc = bitcmp(abs(normc),nbitsabc) + 1;  %Si el numero es negativo se obtiene complemento a 2
      else
        numberc = normc;    %Si el numero es positivo se deja como tal
      end

      biased = Datos_W(j,i) - min_value; %Todos los pesos con valores positivos, para su correcto almacenaje

      normalized_num = round(biased/step);

     count = fprintf(FID2,'\n%s %s %s %s\r', dec2hex(normalized_num), dec2hex(numbera), dec2hex(numberb),
dec2hex(numberc)); %peso y a, b y c
      count = fprintf(FID4,'\n%s\r',dec2hex(normalized_num)); %peso solamente

      count = fprintf(FID_mult(i),'\n%s\r\n%s\r\n%s\r\n%s\r', dec2hex(normalized_num), dec2hex(numbera),
dec2hex(numberb), dec2hex(numberc)); %peso y a, b y c

      count = fprintf(FID_mems(i),'\n%s\r',dec2hex(normalized_num)); %peso solamente

      if (normalized_num_k1 == normalized_num),
        existe_redundancia2 = 1;
      end
      if (j > 2),
       deltas(j-2) = normalized_num - normalized_num_k1;    %se obtiene delta para i-esima memoria
      end
      normalized_num_k1 = normalized_num;
   end
   deltas2 = round(sum(deltas)/length(deltas));
   count = fprintf(FID4,'\n%s\r',dec2hex(deltas2));                %se graba delta promedio solamente, caso memoria
   deltas2_reg(i) = deltas2;
   count = fprintf(FID_mems(i),'\n%s\r',dec2hex(deltas2));      %se graba delta promedio solamente, caso memoria
   clear deltas;
   clear deltas2;
   count = fprintf(FID_mult(i),'\n');
   count = fprintf(FID_mems(i),'\n');
   fclose(FID_mult(i));
   fclose(FID_mems(i));
end

count = fprintf(FID2,'\n');
count = fprintf(FID4,'\n');

fclose(FID2);
fclose(FID4);

%Maximo delta para recorte en verilog
[maxdelta n_mult] = max(deltas2_reg);
maxdelta_hex = dec2hex(maxdelta);

%Obtencion de tamaños de cada multiplicador, obtencion de cantidad de bits
%necesarios para creacion de modulos de memoria ROM
sizes = zeros(columns,4);

for j = 1 : columns,
   sizes(j,1) = 4*Datos_W(1,j) + 1;
```

```
    sizes(j,2) = nextpow2(sizes(j,1));
    sizes(j,3) = 2^(sizes(j,2)) - 1;

    sizes(j,4) = 0;
    if (sizes(j,3) == 255),
        sizes(j,4) = 1;
    end
    sizes(j,2) = sizes(j,2) - 1;
end

%Obtencion de tamaños de cada memoria, obtencion de cantidad de bits
%necesarios para creacion de modulos de memoria ROM
sizes2 = zeros(columns,5);

for j = 1 : columns,
    sizes2(j,1) = Datos_W(1,j) + 2;
    sizes2(j,2) = nextpow2(sizes2(j,1));
    sizes2(j,3) = 2^(sizes2(j,2)) - 1;

    sizes2(j,4) = 0;
    if (sizes2(j,3) == 63),
        sizes2(j,4) = 1;
    end
    sizes2(j,2) = sizes2(j,2) - 1;
    sizes2(j,5) = j;
end
```

A.1.b Creación y prueba de tanh con ecuación de la recta

```
close all
%Tangente hiperbolica por tramos
%Ecuacion de la recta de la forma

% y = m * x + by;

% con m = (y -y0) / (x - x0) y by se obtiene como
% by = y0 - m * x0

bits_m_y_by = 9;            %para direccionar 512 posiciones
width_m_y_by = 16 + 1;   %anchura de 16 bits mas signo
%Memoria de 17 bits x 512 x 2 = 17 bits por 1024 posiciones ==> 17kbits
%(Alcanza en un block RAM)
max_b_value = max(abs(maximob),abs(minimob));
limit = max_b_value;
xsize = 2^(bits_m_y_by);
pasox = xsize;

deltax = limit; %Se cambia delta para barrido solo de valores positivos
xsweep = 0 : deltax/pasox : limit;

%se procede a calcular las pendientes de las rectas
pend = zeros(xsize-1,1);
m = zeros(xsize-1,1);
y0_mx0 = zeros(xsize-1,1);
yy = tanh(xsweep);
```

```
figure
plot(xsweep,yy)
title('Tramo de tangente hiperbolica a convertir')
xlabel('b*x')
ylabel('tanh(b*x) - lado positivo')

delete C:\EmuVerilog\tanh_m.txt
delete C:\EmuVerilog\tanh_b.txt

FID5 = fopen('C:\EmuVerilog\tanh_m.txt','w');
FID6 = fopen('C:\EmuVerilog\tanh_b.txt','w');

if (FID5 == -1)
   error('No se pudo abrir ni crear archivo');
end
if (FID6 == -1)
   error('No se pudo abrir ni crear archivo');
end

for it = 1 : xsize,
  m(it) = (yy(it+1)-yy(it))/(xsweep(it+1) - xsweep(it));
  y0_mx0(it) = yy(it+1) - m(it)*xsweep(it+1);
end

max_m = max(m);  min_m = min(m);
max_y0_mx0 = max(y0_mx0);  min_y0_mx0 = min(y0_mx0);

max_m_y0_mx0 = max(max_m,max_y0_mx0);
min_m_y0_mx0 = min(min_m,min_y0_mx0);

delta_m_y0_mx0 = max_m_y0_mx0 - min_m_y0_mx0;
step_m_y0_mx0 = delta_m_y0_mx0 / (2^(width_m_y_by-1) - 1);

count = fprintf(FID5,'\n@000\r'); %Inicio de datos en memoria
count = fprintf(FID6,'\n@000\r'); %Inicio de datos en memoria
for i = 1 : xsize,

   normalizado_m = round(m(i)/step_m_y0_mx0);
   normalizado_y0_mx0 = round(y0_mx0(i)/step_m_y0_mx0);
   if (normalizado_m < 0),
         number_m = bitcmp(abs(normalizado_m),width_m_y_by) + 1; %Si el numero es negativo se obtiene
                                                                  %complemento a 2
   else
      number_m = normalizado_m;          %Si el numero es positivo se deja como tal
   end

   if (normalizado_y0_mx0 < 0),
      number_y0_mx0 = bitcmp(abs(normalizado_y0_mx0),width_m_y_by) + 1; %Si el numero es negativo se
                                                                        %obtiene complemento a 2
   else
      number_y0_mx0 = normalizado_y0_mx0;          %Si el numero es positivo se deja como tal
   end

   count = fprintf(FID5,'\n%s\r',dec2hex(number_m));
   count = fprintf(FID6,'\n%s\r',dec2hex(number_y0_mx0));
end
```

```
count = fprintf(FID5,'\n');
count = fprintf(FID6,'\n');

fclose(FID5);
fclose(FID6);

%%Reconstruccion de tangente usando rectas
tanh_recovered = zeros(length(xsweep));

for i = 1 : xsize,
   tanh_recovered(i) = m(i)*xsweep(i) + y0_mx0(i);
end
tanh_recovered(i+1) = m(i)*xsweep(i+1) + y0_mx0(i);

% figure
% plot(xsweep,yy)


%Barrido mas extenso para probar interpolaciones
xsw2 = -limit : deltax/(pasox*4) : limit;
xsw2_comp = zeros(1,length(xsw2));

%Las tabulaciones para encontrar m y yo-mxo estan en xsweep

tanh_recovered2 = zeros(length(xsw2),1);

for i = 1 : length(xsw2),
   j = 1;
   dist_winner = 9999999;
   found = 0;

   signo = 0;
   if (xsw2(i) < 0),
      signo = 1; %Si xsw2 es negativo se activa flag de signo
   end

   xsw2_comp(i) = xsw2(i);
   if (signo),
      xsw2_comp(i) = -xsw2_comp(i);
   end

   while (~found),
      dist = abs( xsweep(j) - xsw2_comp(i) );

      if ( dist < dist_winner),
         dist_winner = dist;
         j = j + 1;
         if ( j > 2^(bits_m_y_by) ),
            found = 1;
         end
      else
         found = 1;
      end
   end

   j = j-1;
   tanh_recovered2(i) = m(j)*xsw2_comp(i) + y0_mx0(j);
```

```
   if (signo), %Si el signo es negativo
      tanh_recovered2(i) = -tanh_recovered2(i);
   end

end

figure
plot(xsw2, tanh_recovered2)
title('Tangente hiperbolica reconstruida con interpolacion')
xlabel('b*x')
ylabel('tanh(b*x)')
```

A.1.c Tabulación de puntos para diferentes tanh

```
close all
puntos = 101; %Cantidad de puntos a utilizar (en x) para barrer cada peso
puntosw = 7;  %Cantidad representativa de pesos en multiplicador a utilizar
multiplicador = 20;

deltaw = max_value - min_value;
x2 = -1:2/(puntos-1):1;
ws = 0:65534/(puntosw-1):65534;
wlab = min_value:deltaw/(puntosw-1):max_value;
levels = Datos_W(1,multiplicador);
limit = levels + 1;

pesos = zeros(puntosw,2);
fin = 0;
j = 1;
for i = 1 : puntosw,
  dist_winner = 99999;
  fin = 0;
  while (fin ~= 1),
     dist = abs(Datos_W(j+1,multiplicador) - wlab(i));
     if ( dist < dist_winner),
        dist_winner = dist;
        if ( j == levels ),
           pesos(i,1) = Datos_W(j,multiplicador);
           pesos(i,2) = j-1;
           j = 1;
           fin = 1;
        else
           j = j + 1;
        end
     else
        pesos(i,1) = Datos_W(j,multiplicador);
        pesos(i,2) = j-1;
        j = 1;
        fin = 1;
     end
  end

end

abcs = zeros(puntosw,3);
```

```
for jj = 1 : puntosw,
  abcs(jj,:) = Datos_C(pesos(jj,2),:,multiplicador);
end

max_bx_value = max(abs(minimob),abs(maximob)); %se obtiene maxima magnitud que define la maxima magnitud
                                               %del argumento de tanh
% se tienen todos las constantes utilizadas en la simulacion en verilog
caract = zeros(puntosw,puntos);
for jj = 1 : puntosw,
    a = abcs(jj,1);
    b = abcs(jj,2);
    c = abcs(jj,3);

    bx = round( (b*x2 + max_bx_value)*8191/(2*max_bx_value) ) + 1; %calculo de bx para el barrido de x

    bxmem(jj,:) = bx;
    bx2mem(jj,:) = b*x2;
    bx2 = b*x2;

    tanh2 = zeros(length(bx2),1);
    bx2_comp = zeros(length(bx2),1);

    for i = 1 : length(bx2), %Obtencion de caracteristica para tanh por rectas
      j = 1;
      dist_winner = 9999999;
      found = 0;

      signo = 0;
      if (bx2(i) < 0),
        signo = 1; %Si bx2 es negativo se activa flag de signo
      end

      bx2_comp(i) = bx2(i);
      if (signo),
        bx2_comp(i) = -bx2_comp(i);
      end

      while (~found),
        dist = abs( xsweep(j) - bx2_comp(i) );
        if ( dist < dist_winner),
          dist_winner = dist;
          j = j + 1;
          if ( j > 2^(bits_m_y_by) ),
            found = 1;
          end
        else
          found = 1;
        end
      end

      j = j-1;
      tanh2(i) = m(j)*bx2_comp(i) + y0_mx0(j);
      if (signo), %Si el signo es negativo
        tanh2(i) = -tanh2(i);
      end
    end
```

```
    caract(jj,:) = a*y(bx) + c;
    caract3(jj,:) = a*tanh(bx2) + c;
    caract4(jj,:) = a*tanh2 + c;

end

figure
caract3 = caract3/max(max(caract3));
plot(x2, caract3)
title(sprintf('Caracteristica usando a, b, c y tanh matlab - multiplicador %d',multiplicador))
xlabel('Multiplicando x');
ylabel('Producto w*x');
axis([-1.1 1.1 -1.1 1.1]);
legend('Peso 1','Peso 2','Peso 3','Peso 4','Peso 5','Location','North')

figure
caract4 = caract4 / max(max(caract4));
plot(x2, caract4)
title(sprintf('Caracteristica usando a, b, c y tanh por rectas (interpolacion) - multiplicador %d',multiplicador))
xlabel('Multiplicando x');
ylabel('Producto w*x');
axis([-1.1 1.1 -1.1 1.1]);
legend('Peso 1','Peso 2','Peso 3','Peso 4','Peso 5','Location','North')
```

A.1.d Barrido de tangentes usando datos originales de laboratorio

```
%barrido en x tiene 41 puntos de -1 a 1
close all

sweeps_origen = 41;
xbarr = arrayS_wide{multiplicador}.x(1:sweeps_origen); %cualquier barrido sirve, todos son iguales
lab_source = zeros(puntosw, sweeps_origen);

for ii = 1 : puntosw, %obtencion de curvas desde datos de laboratorio
   startindex = (pesos(ii,2) - 1)*sweeps_origen + 1;
   endindex = startindex + (sweeps_origen-1);
   lab_source(ii,:) = arrayS_wide{multiplicador}.cY(startindex:endindex);
end
lab_source = lab_source / max(max(lab_source));

hventana = figure;
plot(xbarr,lab_source)
axis([-1.1 1.1 -1.1 1.1])
title(sprintf('Caracteristica desde datos originales multiplicador %d',multiplicador))
xlabel('Multiplicando x');
ylabel('Producto w*x');
legend('Peso 1','Peso 2','Peso 3','Peso 4','Peso 5','Location','North')
```

A.1.e Modificación de Verilog para utilización de multiplicadores

```
FID8 = fopen('multiplicador.v.backup','r+t'); %Archivo de respaldo
[FID9, message] = fopen('C:\EmuVerilog\multiplicador.v','w+t'); %Archivo para proyecto

if (FID9 == -1)
```

```
   error(message);
end

multipliers = columns;

foundX = 0;
mult = 5;

[extraccion, ext_count] = fscanf(FID8,'%c',1);
while(ext_count > 0),
   if ( extraccion == 'X'),      %Si se encuentra caracter que indica inicio de creacion de generate
      foundX = 1;
      count = fprintf(FID9,'%c',extraccion);       %Se copia X

      count = fprintf(FID9,'\n\tgenerate');
      count = fprintf(FID9,'\n\t\tcase (nmult)');

      for i = 1 : multipliers,
         count = fprintf(FID9,'\n\t\t\t%d:',i);

         count = fprintf(FID9,' memabc #(.size(%d), .filepath("data_mult%d.txt")) memWabc(index, clk,
abc_memOut);',sizes(i,1),i);

      end
      count = fprintf(FID9,'\n\t\tendcase');
      count = fprintf(FID9,'\n\tendgenerate');
   else
      count = fprintf(FID9,'%c',extraccion);
   end
   [extraccion, ext_count] = fscanf(FID8,'%c',1);
end

fclose(FID8);
fclose(FID9);
```

A.1.f Lectura y modificación de Verilog para utilización de memorias

```
FID10 = fopen('memoria.v.backup','r+t'); %Archivo de respaldo
[FID11, message2] = fopen('C:\EmuVerilog\memoria.v','w+t'); %Archivo para proyecto

if (FID11 == -1)
   error(message2);
end

memories = columns;

foundX = 0;

[extraccion, ext_count] = fscanf(FID10,'%c',1);
while(ext_count > 0),
   if ( (extraccion == 'X') && (foundX == 0) ),      %Si se encuentra caracter que indica inicio de creacion de generate
      foundX = 1;
      count = fprintf(FID11,'%c',extraccion);     %Copia X encontrado

      count = fprintf(FID11,'\n\tgenerate');
```

```
        count = fprintf(FID11,'\n\t\tcase (memory)');

        for i = 1 : memories,
            count = fprintf(FID11,'\n\t\t\t%d:',i);

            count = fprintf(FID11,' memmem #(.size(%d), .filepath("data_mem%d.txt")) tablamemW(index, clk,
memOut);',sizes2(i,1),i);
        end
        count = fprintf(FID11,'\n\t\tendcase');
        count = fprintf(FID11,'\n\tendgenerate');
    else
        count = fprintf(FID11,'%c',extraccion);
    end
    [extraccion, ext_count] = fscanf(FID10,'%c',1);
end

fclose(FID10);
fclose(FID11);
```

A.1.g Generación de verilog para creación de red neuronal

```
clc
its_ok = 0;
while (~its_ok),
    its_ok = 1;
    sinapsis = input(sprintf('Ingrese la(s) sinapsis (de 1 a 64) para generacion de perceptron,\n si ingresa mas de una
escriba dentro de []: '));
    sinaps_totales = length(sinapsis);
    for i = 1 : sinaps_totales,
      if( (sinapsis(i) <= 0) || (sinapsis(i) > 64)),
          its_ok = 0;
          disp(sprintf('\nIngrese valores dentro del rango permisible de 1 a 64'));
      end
    end
end

puntos = input('Ingrese la cantidad de puntos a presentar a la red (ej. 1024): ');

rand('state', 0);
fixed_weigths = round(rand(sinaps_totales,1)*(2^16 - 1));

valores_iniciales = zeros(sinaps_totales,1);
disp(sprintf('Valores iniciales y pesos fijos escogidos al azar para cada sinapsis:\n'))
for i = 1 : sinaps_totales,
    valores_iniciales(i) = round(rand()*(Datos_W(1,sinapsis(i))-1) + 1);
    disp(sprintf('Sinapsis %i\t- \tNivel inicial = %i\t\tValor fijo red referencia =
%i',sinapsis(i),valores_iniciales(i),fixed_weigths(i)));
end

%% Apertura y modificacion de archivo perceptron.v.backup
 FID12 = fopen('perceptron.v.backup','r+t'); %Archivo de respaldo
[FID13, message3] = fopen('C:\EmuVerilog\perceptron.v','w+t'); %Archivo para proyecto

if (FID13 == -1)
    error(message3);
```

```
end

foundX1 = 0; foundX2 = 0; foundX3 = 0; foundX4 = 0; foundX5 = 0; foundX6 = 0; foundX7 = 0; foundX8 = 0;
foundX9 = 0;

[extraccion, ext_count] = fscanf(FID12,'%c',1);
while(ext_count > 0),
  if ( (extraccion == 'X') && (foundX1 == 0) ),      %Si se encuentra caracter que indica inicio de creacion de codigo
    foundX1 = 1;
    count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

    count = fprintf(FID13,'\nmodule perceptron(');

    for i = 1 : sinaps_totales,
      count = fprintf(FID13,'x%i, dw%i, ',i,i);
    end

    count = fprintf(FID13,'clk, startmults, startmems, readymems, readymults, Y');

    for i = 1 : sinaps_totales,
      count = fprintf(FID13,', new_weight%i, oldremainder%i',i,i);
    end
     count = fprintf(FID13,');\n\n');

    for i = 1 : sinaps_totales,
      count = fprintf(FID13,'\tinput signed [16:0] x%i;\n\tinput signed [17:0] dw%i;\n',i,i);
    end

  elseif ( (extraccion == 'X') && (foundX9 == 0) ),
    foundX9 = 1;
    count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

    for i = 1 : sinaps_totales,
      count = fprintf(FID13,'\n\toutput [15:0] new_weight%i;',i);
      count = fprintf(FID13,'\n\toutput signed [17:0] oldremainder%i;',i);
    end

  elseif ( (extraccion == 'X') && (foundX2 == 0) ),
    foundX2 = 1;
    count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

    for i = 1 : sinaps_totales,
      count = fprintf(FID13,'\n\t//----------------------------------//');
      count = fprintf(FID13,'\n\t//Variables sinapsis %i',sinapsis(i));
      count = fprintf(FID13,'\n\treg start_mult%i = 0;',i);
      count = fprintf(FID13,'\n\treg start_mem%i = 0;',i);
      count = fprintf(FID13,'\n\treg readyme_%i_up;',i);
      count = fprintf(FID13,'\n\treg readymu_%i_up;',i);

      count = fprintf(FID13,'\n\twire signed [17:0] y%i;',i);
      count = fprintf(FID13,'\n\twire ready_mem%i;',i);
      count = fprintf(FID13,'\n\twire ready_mult%i;',i);
    end

  elseif ( (extraccion == 'X') && (foundX3 == 0) ),
    foundX3 = 1;
    count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado
```

```
        for i = 1 : sinaps_totales,
            count = fprintf(FID13,'\n\tsinapsis #(.numero(%i),.initial_index(%i)) synapse%i(dw%i, x%i, clk, start_mult%i,
start_mem%i, y%i, ready_mem%i, ready_mult%i, new_weight%i,
oldremainder%i);',sinapsis(i),valores_iniciales(i),i,i,i,i,i,i,i,i,i,i);
        end

    elseif ( (extraccion == 'X') && (foundX4 == 0) ),
        foundX4 = 1;
        count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

        for i = 1 : sinaps_totales,
            count = fprintf(FID13,'\n\t\tif (~ready_mem%i) start_mem%i <= 0;',i,i);
            count = fprintf(FID13,'\n\t\tif (~ready_mult%i) start_mult%i <= 0;',i,i);
        end
    elseif ( (extraccion == 'X') && (foundX5 == 0) ),
        foundX5 = 1;
        count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

        for i = 1 : sinaps_totales,
            count = fprintf(FID13,'\n\n\t\t\t\t\treadyme_%i_up <= 0;',i);
            count = fprintf(FID13,'\n\t\t\t\t\tstart_mem%i <= 1;',i);
        end
        count = fprintf(FID13,'\n\n\t\t\t\t\tmem_sinapsis_counter <= %i;',sinaps_totales);
    elseif ( (extraccion == 'X') && (foundX6 == 0) ),
        foundX6 = 1;
        count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

        for i = 1 : sinaps_totales,
            if (i > 1),
                count = fprintf(FID13,'\n\t\t\t\telse if (ready_mem%i && ~readyme_%i_up) begin',i,i);
            else
                count = fprintf(FID13,'\n\t\t\t\tif (ready_mem%i && ~readyme_%i_up) begin',i,i);
            end
            count = fprintf(FID13,'\n\t\t\t\t\tmem_sinapsis_counter <= mem_sinapsis_counter - 1;');
            count = fprintf(FID13,'\n\t\t\t\t\treadyme_%i_up <= 1;',i);
            count = fprintf(FID13,'\n\t\t\t\tend');
        end
    elseif ( (extraccion == 'X') && (foundX7 == 0) ),
        foundX7 = 1;
        count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

        for i = 1 : sinaps_totales,
            count = fprintf(FID13,'\n\n\t\t\t\t\treadymu_%i_up <= 0;',i);
            count = fprintf(FID13,'\n\t\t\t\t\tstart_mult%i <= 1;',i);
        end
        count = fprintf(FID13,'\n\n\t\t\t\t\tmul_sinapsis_counter <= %i;',sinaps_totales);
    elseif ( (extraccion == 'X') && (foundX8 == 0) ),
        foundX8 = 1;
        count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

        for i = 1 : sinaps_totales,
            if (i > 1),
                count = fprintf(FID13,'\n\t\t\t\telse if (ready_mult%i && ~readymu_%i_up) begin',i,i);
            else
                count = fprintf(FID13,'\n\t\t\t\tif (ready_mult%i && ~readymu_%i_up) begin',i,i);
            end
```

```
            count = fprintf(FID13,'\n\t\t\t\t\tmul_sinapsis_counter <= mul_sinapsis_counter - 1;');
            count = fprintf(FID13,'\n\t\t\t\t\tY <= Y + y%i;',i);
            count = fprintf(FID13,'\n\t\t\t\t\treadymu_%i_up <= 1;',i);
            count = fprintf(FID13,'\n\t\t\t\tend');
        end
    else
        count = fprintf(FID13,'%c',extraccion);
    end
    [extraccion, ext_count] = fscanf(FID12,'%c',1);
end

fclose(FID12);
fclose(FID13);

%% Apertura y modificacion de archivo red_neuronal.v.backup
clear FID12;
clear FID13;
FID12 = fopen('red_neuronal.v.backup','r+t'); %Archivo de respaldo
[FID13, message4] = fopen('C:\EmuVerilog\red_neuronal.v','w+t'); %Archivo para proyecto

if (FID13 == -1)
    error(message4);
end

foundX1 = 0; foundX2 = 0; foundX3 = 0; foundX4 = 0; foundX5 = 0; foundX6 = 0; foundX7 = 0; foundX8 = 0;
foundX9 = 0; foundX10 = 0;

[extraccion, ext_count] = fscanf(FID12,'%c',1);
while(ext_count > 0),
    if ( (extraccion == 'X') && (foundX1 == 0) ),      %Si se encuentra caracter que indica inicio de creacion de codigo
        foundX1 = 1;
        count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

        count = fprintf(FID13,'\nmodule red_neuronal(');

        for i = 1 : sinaps_totales,
            count = fprintf(FID13,'x%i, ',i);
        end

        count = fprintf(FID13,'start, mode, clk, ready, error');

        for i = 1 : sinaps_totales,
            count = fprintf(FID13,', new_weight%i, oldremainder%i',i,i);
        end
        count = fprintf(FID13,', d);\n\n');

        for i = 1 : sinaps_totales,
            count = fprintf(FID13,'\tinput signed [16:0] x%i;\n',i);
        end

    elseif ( (extraccion == 'X') && (foundX2 == 0) ),
        foundX2 = 1;
        count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

        for i = 1 : sinaps_totales,
            count = fprintf(FID13,'\n\toutput [15:0] new_weight%i;',i);
            count = fprintf(FID13,'\n\toutput signed [17:0] oldremainder%i;',i);
```

```
    end

  elseif ( (extraccion == 'X') && (foundX3 == 0) ),
    foundX3 = 1;
    count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

    for i = 1 : sinaps_totales,
      count = fprintf(FID13,'\n\twire signed [10:0] x%inu = x%i[16:6];',i,i);
      count = fprintf(FID13,'\n\twire signed [32:0] x%inu_error = x%inu*recorte_error;',i,i);
      count = fprintf(FID13,'\n\twire signed [16:0] recorte_delta%i = x%inu_error[31:15];',i,i);
      count = fprintf(FID13,'\n\twire signed [24:0] reescalada_delta%i = recorte_delta%i*constante;',i,i);
      count = fprintf(FID13,'\n\twire signed [16:0] delta%i_final = reescalada_delta%i[23:7];\n',i,i);
    end

  elseif ( (extraccion == 'X') && (foundX4 == 0) ),
    foundX4 = 1;
    count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

    for i = 1 : sinaps_totales,
      count = fprintf(FID13,'\n\treg signed [17:0] dw%i = 0;',i);
    end

  elseif ( (extraccion == 'X') && (foundX5 == 0) ),
    foundX5 = 1;
    count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

    for i = 1 : sinaps_totales,
      count = fprintf(FID13,'\n\t//Variables multiplicador %i de peso fijo',i);
      count = fprintf(FID13,'\n\treg startm%i = 0;',i);
      count = fprintf(FID13,'\n\treg readymu_%i_up;',i);
      count = fprintf(FID13,'\n\twire signed [17:0] wx%i;',i);
      count = fprintf(FID13,'\n\twire readym%i;\n',i);
    end
  elseif ( (extraccion == 'X') && (foundX6 == 0) ),
    foundX6 = 1;
    count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

    count = fprintf(FID13,'\n\tperceptron per1(');

    for i = 1 : sinaps_totales,
      count = fprintf(FID13,'x%i, dw%i, ',i,i);
    end
    count = fprintf(FID13,'clk, startmults, startmems, readymems, readymults, Y');

    for i = 1 : sinaps_totales,
      count = fprintf(FID13,', new_weight%i, oldremainder%i',i,i);
    end
    count = fprintf(FID13,');\n');

    for i = 1 : sinaps_totales,
      count = fprintf(FID13,'\n\tmultiplicador2 #(.nmult(%i)) m%i(startm%i, clk, x%i, 16''h%X, wx%i,
readym%i);',sinapsis(i),i,i,i,fixed_weigths(i),i,i);
    end
  elseif ( (extraccion == 'X') && (foundX7 == 0) ),
    foundX7 = 1;
    count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado
```

```
      for i = 1 : sinaps_totales,
         count = fprintf(FID13,'\n\t\tif (~readym%i) startm%i <= 0;',i,i);
      end
   elseif ( (extraccion == 'X') && (foundX8 == 0) ),
      foundX8 = 1;
      count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

      count = fprintf(FID13,'\n\t\t\t\t\t//arranque multiplicadores de pesos fijos');

      for i = 1 : sinaps_totales,
         count = fprintf(FID13,'\n\t\t\t\tstartm%i <= 1;',i);
         count = fprintf(FID13,'\n\t\t\t\treadymu_%i_up <= 0;\n',i);
      end
      count = fprintf(FID13,'\n\t\t\t\tmul_sinapsis_counter <= %i;',sinaps_totales);
   elseif ( (extraccion == 'X') && (foundX9 == 0) ),
      foundX9 = 1;
      count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

      for i = 1 : sinaps_totales,
         if (i > 1),
            count = fprintf(FID13,'\n\t\t\telse if (readym%i && ~readymu_%i_up) begin',i,i);
         else
            count = fprintf(FID13,'\n\t\t\tif (readym%i && ~readymu_%i_up) begin',i,i);
         end
         count = fprintf(FID13,'\n\t\t\t\tmul_sinapsis_counter <= mul_sinapsis_counter - 1;');
         count = fprintf(FID13,'\n\t\t\t\td <= d + wx%i;',i);
         count = fprintf(FID13,'\n\t\t\t\treadymu_%i_up <= 1;',i);
         count = fprintf(FID13,'\n\t\t\tend');
      end
   elseif ( (extraccion == 'X') && (foundX10 == 0) ),
      foundX10 = 1;
      count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

      for i = 1 : sinaps_totales,
         count = fprintf(FID13,'\n\t\t\tdw%i <= delta%i_final;',i,i);
      end
   else
      count = fprintf(FID13,'%c',extraccion);
   end
   [extraccion, ext_count] = fscanf(FID12,'%c',1);
end

fclose(FID12);
fclose(FID13);

%% Generacion de archivo de simulacion verilog red_neuronal_testbench.v
clear FID12;
clear FID13;
FID12 = fopen('red_neuronal_testbench.v.backup','r+t'); %Archivo de respaldo
[FID13, message4] = fopen('C:\EmuVerilog\red_neuronal_testbench.v','w+t'); %Archivo para proyecto

if (FID13 == -1)
   error(message4);
end

foundX1 = 0; foundX2 = 0; foundX3 = 0;
```

```
[extraccion, ext_count] = fscanf(FID12,'%c',1);
while (ext_count > 0),
   if ( (extraccion == 'X') && (foundX1 == 0) ),     %Si se encuentra caracter que indica inicio de creacion de codigo
      foundX1 = 1;
      count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

      for i = 1 : sinaps_totales,
         count = fprintf(FID13,'\n\treg signed [16:0] xx%i[0:%i];',i,puntos-1);
         count = fprintf(FID13,'\n\treg signed [16:0] x%i;',i);
         count = fprintf(FID13,'\n\twire [15:0] new_weight%i;',i);
         count = fprintf(FID13,'\n\tinteger handle%i;',i);
         count = fprintf(FID13,'\n\twire signed [17:0] oldremainder%i;\n',i);
      end
   elseif ( (extraccion == 'X') && (foundX2 == 0) ),
      foundX2 = 1;
      count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

      count = fprintf(FID13,'\n\tred_neuronal test(');

      for i = 1 : sinaps_totales,
         count = fprintf(FID13,'x%i, ',i);
      end

      count = fprintf(FID13,'start, mode, clk, ready, error');

      for i = 1 : sinaps_totales,
         count = fprintf(FID13,', new_weight%i, oldremainder%i',i,i);
      end

      count = fprintf(FID13,', d);\n');

      count = fprintf(FID13,'\n\tinitial begin');

      for i = 1 : sinaps_totales,
         count = fprintf(FID13,'\n\t\t$readmemh("x%i.txt", xx%i);\n\t\thandle%i = $fopen("peso%i.txt");\n',i,i,i,i);
      end

      count = fprintf(FID13,'\n\t\thandle_error = $fopen("error.txt");\n');
      count = fprintf(FID13,'\tend\n');
      count = fprintf(FID13,'\n\talways @(posedge ready) begin');

      for i = 1 : sinaps_totales,
         count = fprintf(FID13,'\n\t\t$fdisplay(handle%i,"%%d",new_weight%i);',i,i);
      end

      count = fprintf(FID13,'\n\n\t\tif(error_counter > 0) begin');
      count = fprintf(FID13,'\n\t\t\t$fdisplay(handle_error,"%%d",error);');
      count = fprintf(FID13,'\n\t\tend');
      count = fprintf(FID13,'\n\n\terror_counter = error_counter + 1;');
      count = fprintf(FID13,'\n\tend\n');

   elseif ( (extraccion == 'X') && (foundX3 == 0) ),
      foundX3 = 1;
      count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado
      count = fprintf(FID13,'\n\t\tif (counter != %i) begin',puntos-1);

      for i = 1 : sinaps_totales,
```

```
        count = fprintf(FID13,'\n\t\t\t\tx%i <= xx%i[counter];',i,i);
      end
   else
      count = fprintf(FID13,'%c',extraccion);
   end
   [extraccion, ext_count] = fscanf(FID12,'%c',1);
end

fclose(FID12);
fclose(FID13);

% Generacion de archivo de simulacion verilog testfinal_FPGA.v
clear FID12;
clear FID13;

FID12 = fopen('testfinal_FPGA.v.backup','r+t'); %Archivo de respaldo
[FID13, message4] = fopen('C:\EmuVerilog\testfinal_FPGA.v','w+t'); %Archivo para proyecto

if (FID13 == -1)
   error(message4);
end

foundX1 = 0; foundX2 = 0; foundX3 = 0; foundX4 = 0;

[extraccion, ext_count] = fscanf(FID12,'%c',1);
while (ext_count > 0),
   if ( (extraccion == 'X') && (foundX1 == 0) ),     %Si se encuentra caracter que indica inicio de creacion de codigo
      foundX1 = 1;
      count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

      for i = 1 : sinaps_totales,
         count = fprintf(FID13,'\n\twire signed [16:0] xx%i;',i);
         count = fprintf(FID13,'\n\treg signed [16:0] x%i = %i;',i,i);
         count = fprintf(FID13,'\n\twire [15:0] new_weight%i;',i);
         count = fprintf(FID13,'\n\twire signed [17:0] oldremainder%i;\n',i);
      end
   elseif ( (extraccion == 'X') && (foundX2 == 0) ),
      foundX2 = 1;
      count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

      count = fprintf(FID13,'\n\tred_neuronal test(');

      for i = 1 : sinaps_totales,
         count = fprintf(FID13,'x%i, ',i);
      end

      count = fprintf(FID13,'start, mode, clk, ready, error');

      for i = 1 : sinaps_totales,
         count = fprintf(FID13,', new_weight%i, oldremainder%i',i,i);
      end

      count = fprintf(FID13,', d);\n');

      count = fprintf(FID13,'\n\t//memoria para x''s');

      for i = 1 : sinaps_totales,
```

```
        count = fprintf(FID13,'\n\tmem_xs #(.size(%i),.filepath("x%i.txt")) X%i_data(counter, clk, xx%i);',puntos,i,i,i);
    end

elseif ( (extraccion == 'X') && (foundX3 == 0) ),
    foundX3 = 1;
    count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

    for i = 1 : sinaps_totales,
        count = fprintf(FID13,'\n\t\t\t\tx%i <= xx%i;',i,i);
    end
elseif ( (extraccion == 'X') && (foundX4 == 0) ),
    foundX4 = 1;
    state = 2;
    count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

    for i = 1 : sinaps_totales,
        count = fprintf(FID13,'\n\t\t\t\t\t%i: begin',state);

        count = fprintf(FID13,'\n\t\t\t\t\t\tASCII_msg <= "new_weight%i";',i);
        count = fprintf(FID13,'\n\t\t\t\t\t\tnumber <= new_weight%i;',i);
        count = fprintf(FID13,'\n\t\t\t\t\t\tstart_LCD <= 1;');
        count = fprintf(FID13,'\n\t\t\t\t\t\tLCD_state <= LCD_state + 1;');

        count = fprintf(FID13,'\n\t\t\t\t\tend');

        state = state + 1;

        count = fprintf(FID13,'\n\t\t\t\t\t%i: begin',state);

        count = fprintf(FID13,'\n\t\t\t\t\t\tASCII_msg <= "oldremainder%i";',i);
        count = fprintf(FID13,'\n\t\t\t\t\t\tnumber <= oldremainder%i;',i);
        count = fprintf(FID13,'\n\t\t\t\t\t\tstart_LCD <= 1;');
        count = fprintf(FID13,'\n\t\t\t\t\t\tLCD_state <= LCD_state + 1;');

        count = fprintf(FID13,'\n\t\t\t\t\tend');

        state = state + 1;

        count = fprintf(FID13,'\n\t\t\t\t\t%i: begin',state);

        count = fprintf(FID13,'\n\t\t\t\t\t\tASCII_msg <= "x%i";',i);
        count = fprintf(FID13,'\n\t\t\t\t\t\tnumber <= x%i;',i);
        count = fprintf(FID13,'\n\t\t\t\t\t\tstart_LCD <= 1;');
        if (i == sinaps_totales),
            count = fprintf(FID13,'\n\t\t\t\t\t\tLCD_state <= 0;');
        else
            count = fprintf(FID13,'\n\t\t\t\t\t\tLCD_state <= LCD_state + 1;');
        end
        count = fprintf(FID13,'\n\t\t\t\t\tend');

        state = state + 1;
    end

else
    count = fprintf(FID13,'%c',extraccion);
end
[extraccion, ext_count] = fscanf(FID12,'%c',1);
```

```
end

fclose(FID12);
fclose(FID13);

%% Generacion de archivo de simulacion verilog emu_serial.v
clear FID12;
clear FID13;
FID12 = fopen('emu_serial.v.backup','r+t'); %Archivo de respaldo
[FID13, message4] = fopen('C:\EmuVerilog\emu_serial.v','w+t'); %Archivo para proyecto

if (FID13 == -1)
   error(message4);
end

foundX1 = 0; foundX2 = 0; foundX3 = 0; foundX4 = 0;

[extraccion, ext_count] = fscanf(FID12,'%c',1);
while (ext_count > 0),
   if ( (extraccion == 'X') && (foundX1 == 0) ),      %Si se encuentra caracter que indica inicio de creacion de codigo
      foundX1 = 1;
      count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

      for i = 1 : sinaps_totales,
         count = fprintf(FID13,'\n\twire signed [16:0] xx%i;',i);
         count = fprintf(FID13,'\n\treg signed [16:0] x%i = %i;',i,i);
         count = fprintf(FID13,'\n\twire [15:0] new_weight%i;',i);
         count = fprintf(FID13,'\n\twire signed [17:0] oldremainder%i;\n',i);
      end

      count = fprintf(FID13,'\n\treg [10:0] data_points = %i; //puntos-1\n',puntos-1);
   elseif ( (extraccion == 'X') && (foundX2 == 0) ),
      foundX2 = 1;
      count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

      count = fprintf(FID13,'\n\tred_neuronal test(');

      for i = 1 : sinaps_totales,
         count = fprintf(FID13,'x%i, ',i);
      end

      count = fprintf(FID13,'start, mode, clk, ready, error');

      for i = 1 : sinaps_totales,
         count = fprintf(FID13,', new_weight%i, oldremainder%i',i,i);
      end

      count = fprintf(FID13,', d);\n');

      count = fprintf(FID13,'\n\t//memoria para x"s');

      for i = 1 : sinaps_totales,
         count = fprintf(FID13,'\n\tmem_xs #(.size(%i),.filepath("x%i.txt")) X%i_data(counter, clk, xx%i);',puntos,i,i,i);
      end

   elseif ( (extraccion == 'X') && (foundX3 == 0) ),
      foundX3 = 1;
```

```
      count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

      for i = 1 : sinaps_totales,
         count = fprintf(FID13,'\n\t\t\t\tx%i <= xx%i;',i,i);
      end
   elseif ( (extraccion == 'X') && (foundX4 == 0) ),
      foundX4 = 1;
      state = 2;
      count = fprintf(FID13,'%c',extraccion);     %Copia X encontrado

      for i = 1 : sinaps_totales,
         count = fprintf(FID13,'\n\t\t\t%i: begin',state);

         if (i == 1),
            count = fprintf(FID13,'\n\t\t\t\tif (ready) begin');
         else
            count = fprintf(FID13,'\n\t\t\t\tif (ready_sending) begin');
         end

         count = fprintf(FID13,'\n\t\t\t\t\tnumber <= new_weight%i;',i);
         count = fprintf(FID13,'\n\t\t\t\t\tstart_sending <= 1;');
         count = fprintf(FID13,'\n\t\t\t\t\tstate <= state + 1;');

         count = fprintf(FID13,'\n\t\t\t\tend');
         count = fprintf(FID13,'\n\t\t\tend');

         state = state + 1;
      end

      count = fprintf(FID13,'\n\t\t\t%i: begin',state);
      count = fprintf(FID13,'\n\t\t\t\tif (ready_sending) begin');
      count = fprintf(FID13,'\n\t\t\t\t\tstate <= nextstate;');
      count = fprintf(FID13,'\n\t\t\t\tend');
      count = fprintf(FID13,'\n\t\t\tend');

   else
      count = fprintf(FID13,'%c',extraccion);
   end
   [extraccion, ext_count] = fscanf(FID12,'%c',1);
end

fclose(FID12);
fclose(FID13);

%%  Numeros aleatorios para vector de entrada
FID_xs = zeros(sinaps_totales,1);
xramdom = zeros(puntos,sinaps_totales+1);
rand('state', 0);

for i = 1 : 64,
   existencia = exist(sprintf('C:\\EmuVerilog\\x%i.txt',i),'file');
   if (existencia), delete(sprintf('C:\\EmuVerilog\\x%i.txt',i)); end
end

for i = 1 : sinaps_totales,
   FID_xs(i) = fopen(sprintf('C:\\EmuVerilog\\x%i.txt',i),'w');
   count = fprintf(FID_xs(i),'\n@000\r');              %Inicio de datos en memoria
```

```
   xramdom(:,i) = round((rand(puntos,1)-0.5)*2*(2^16 - 2) + 1);
end
xramdom(:,i+1) = (0:puntos-1)';

bits = 16 + 1; % -65536  a 65535

for i = 1 : sinaps_totales,
   for j = 1 : puntos,
      if (xramdom(j,i) >= 0),
         count = fprintf(FID_xs(i),'\n%s\r',dec2hex(xramdom(j,i)));
      else
         count = fprintf(FID_xs(i),'\n%s\r',dec2hex(bitcmp(abs(xramdom(j,i)),bits)+1));
      end
   end
   count = fprintf(FID_xs(i),'\n');
   fclose(FID_xs(i));
end
```

A.1.h Resultados desde ISE™ Webpack™ para multiplicador

```
% Operacion:
% multiplicacion = a*tanh(b*x) + c    - parametros a, b y c dependen de
% peso w
% x barre desde -65534 a 65534 en xx puntos
% w se barre desde 0 (valor mas negativo) hasta FFFF (valor mas positivo),
% en 7 puntos

clear FID3
FID3 = fopen('c:\EmuVerilog\curva.txt','r');

tline = fgets(FID3);
multiplicador = str2double(tline); %Se obtiene numero de multiplicador
puntos = 101;

x = -1:2/(puntos-1):1;

xx = round(x*65535)';

for i = 1 : puntos-1,
  cambio(i) = xx(i+1) - xx(i);
end
cambiomedio = round(sum(cambio)/length(cambio));

xverilog = xx';
caracteristica3 = zeros(7,101);

for ij = 1 : 7,
   for kk = 1 : 101,
      tline = fgets(FID3);
      number = str2double(tline);

      caracteristica3(ij,kk) = number;
   end
end
xverilog = xverilog/max(xverilog);
```

```
caracteristica3 = caracteristica3/max(max(caracteristica3));

figure
plot(xverilog,caracteristica3)
title(sprintf('Caracteristica desde verilog multiplicador %d',multiplicador))
axis([-1.1 1.1 -1.1 1.1]);

fclose(FID3);
```

A.1.i Extracción de datos de emulación de multiplicador en FPGA

```
%Datos de entrada de 18 bits, en 6 numeros hex (maximo 5)
%cuando numero hex es mayor que 02 00 00
Threshold = hex2dec('20000');

bits = 17 + 1;
puntos = 101;
curvas = 7;
x_reconstructed = -1 : 2/(puntos-1) : 1;

clear FID7
FID7 = fopen('C:\Users\D@nnyel\Documents\UdeC 2007\2007 - II\Memoria de
Titulo\DevSerialTest\resultadosFPGA.txt','r');

multiplicador = 64;

caracteristica4 = zeros(curvas,puntos);

for ij = 1 : curvas,
   for kk = 1 : puntos,
      tline = fgets(FID7);
      number = str2double(tline);

      caracteristica4(ij,kk) = number;
   end
end
xverilog = x_reconstructed;
caracteristica4 = caracteristica4/max(max(caracteristica4));

figure
plot(xverilog,caracteristica4)
title(sprintf('Caracteristica desde FPGA multiplicador %d',multiplicador))
axis([-1.1 1.1 -1.1 1.1]);

fclose(FID7);
```

A.1.j Extracción de datos de simulación red neuronal

```
FID_weight = zeros(sinaps_totales,1);
factor_normalizacion = 65535;

for i = 1 : sinaps_totales,
   FID_weight(i) = fopen(sprintf('C:\\EmuVerilog\\peso%i.txt',i),'r');
end
```

```
weights = zeros(puntos,sinaps_totales);

stringg2 ='';
for i = 1 : sinaps_totales,
   if (i == sinaps_totales),
      stringnew2 = '%i ';
   elseif (i == sinaps_totales-1)
      stringnew2 = '%i y ';
   else
      stringnew2 = '%i, ';
   end
   stringg2 = [stringg2 stringnew2];
end

for i = 1 : sinaps_totales,
   for j = 1 : puntos,
      tline = fgets(FID_weight(i));
      number = str2double(tline);
      weights(j,i) = number;
   end
   fclose(FID_weight(i));
end

weights = weights / factor_normalizacion; %Normalizacion de resultados
figure
plot(weights)
axis([0 puntos 0 65535/factor_normalizacion])
titulo = sprintf('Evolucion pesos de sinapsis %s - Simulacion verilog',stringg2);
title(sprintf(titulo,sinapsis))
```

A.1.k Extracción de datos de emulación de red neuronal en FPGA

```
FID_weight = zeros(sinaps_totales,1);
factor_normalizacion = 65535;

clear FID
FID = fopen('C:\Users\D@nnyel\Documents\UdeC 2007\2007 - II\Memoria de Titulo\DevSerialTest\emuFPGA.txt','r');
weights2 = zeros(sinaps_totales,puntos);

for i = 1 : puntos,
   for j = 1 : sinaps_totales,
      tline = fgets(FID);
      number = str2double(tline);
      weights2(j,i) = number;
   end
end

cadena = '';
stringg2 ='';
for i = 1 : sinaps_totales,
   if (i == sinaps_totales),
      stringnew2 = '%i ';
   elseif (i == sinaps_totales-1)
      stringnew2 = '%i y ';
```

```
    else
        stringnew2 = '%i, ';
    end
    stringg2 = [stringg2 stringnew2];
    if (sinapsis(i)<10),
        cadena(i,:) = sprintf('Sinapsis 0%i',sinapsis(i));
    else
        cadena(i,:) = sprintf('Sinapsis %i',sinapsis(i));
    end
end

weights2 = weights2 / factor_normalizacion; %Normalizacion de resultados
ventana = figure;
plot(weights2')
axis([0 puntos 0 65535/factor_normalizacion])
titulo = sprintf('Evolucion pesos de sinapsis %s - Emulacion',stringg2);
title(sprintf(titulo,sinapsis))
xlabel('Iteraciones')
ylabel('Valores de pesos')
legend(cadena,'Location','NorthEastOutside')
set(ventana,'Position',[100 79 650 420])
fclose(FID);
```

A.2 Módulos Verilog HDL

El código fuente de varios de los archivos expuestos en esta sección, corresponden a generaciones mediante script en Matlab®, para efectos de evitar códigos extensos, se ingresó en linea de comando, la generación de sólo dos sinápsis.

A.2.a Emulación red neuronal y envio de resultados por puerto serial

```
`timescale 1ns / 1ps
module emu_serial(Dbtn0, clk_unbuffered, TxD_serial);

        input Dbtn0;
        input clk_unbuffered;

        output TxD_serial;

        reg backdown = 0;

        wire clklow;
        wire btn0;

        //Variables generadas matlab
        // Memorias para presentacion a la red -- 1X
        wire signed [16:0] xx1;
        reg signed [16:0] x1 = 1;
        wire [15:0] new_weight1;
        wire signed [17:0] oldremainder1;
```

```
	wire signed [16:0] xx2;
	reg signed [16:0] x2 = 2;
	wire [15:0] new_weight2;
	wire signed [17:0] oldremainder2;

	reg [10:0] data_points = 1023; //puntos-1

	wire signed [35:0] d;
	reg [9:0] counter = 0;

	reg start = 0;
	reg mode; //Modo de red, 0 = sin entrenamiento, 1 = con actualizacion de pesos

	wire ready;
	wire [35:0] error;

	wire clkbuf, clk0, clk;

	wire signed [35:0] Y;

	reg [3:0] state = 0;
	reg [3:0] nextstate = 0;
	//--- termino generacion variables matlab para declaraciones

	BUFGDLL clkbuffer(.O(clkbuf), .I(clk_unbuffered));

	clk_reductor reduceclk_dbtns(clkbuf,clklow);
	antidebouncing ADB0(clklow, Dbtn0, btn0);

	clkdiv2 reduceamitad0(clkbuf, clk);

	//Vars envia numero via serial
	reg [17:0] number;
	reg start_sending = 0;
	wire ready_sending;

	send_number_serial envia(number, clk, start_sending, ready_sending, TxD_serial);

	// declaracion de red neuronal -- 2X
	red_neuronal test(x1, x2, start, mode, clk, ready, error, new_weight1, oldremainder1, new_weight2,
oldremainder2, d);

	//memoria para x's
	mem_xs #(.size(1024),.filepath("x1.txt")) X1_data(counter, clk, xx1);
	mem_xs #(.size(1024),.filepath("x2.txt")) X2_data(counter, clk, xx2);

	always  @(negedge clk) begin
		if (~ready) start <= 0;
		if (~ready_sending) start_sending <= 0;

		case (state)
			0: begin
				if (ready && btn0 && backdown) begin
					backdown <= 0;
					counter <= 0;

					nextstate <= 1;
```

```
                                        state <= state + 1;
                                end
                                else if (~btn0) begin
                                        backdown <= 1;
                                end
                        end
                        1: begin // -- 3X
                                x1 <= xx1;
                                x2 <= xx2;

                                start <= 1;             //inicio excitacion red neuronal

                                if (counter < data_points) counter <= counter + 1;      //se apunta en siguiente --
                                                                                        //espacio de memoria
                                else if (counter == data_points) begin
                                        nextstate <= 0;     //se termina luego de enviar ultimos resultados
                                end

                                state <= state + 1;
                        end // -- 4X
                        2: begin
                                if (ready) begin
                                        number <= new_weight1;
                                        start_sending <= 1;
                                        state <= state + 1;
                                end
                        end
                        3: begin
                                if (ready_sending) begin
                                        number <= new_weight2;
                                        start_sending <= 1;
                                        state <= state + 1;
                                end
                        end
                        4: begin
                                if (ready_sending) begin
                                        state <= nextstate;
                                end
                        end
                        default: begin
                                state <= 0;
                        end
                endcase
        end //fin always

endmodule
```

A.2.b Reductor de frecuencia para anti-rebote

```
`timescale 1ns / 1ps
module clk_reductor(clk,clklow);
        input clk;
        output reg clklow = 0;
```

```
	reg [13:0] counter = 0;

	always  @(posedge clk) begin
		if (clklow) begin
			counter <= 0;
			clklow <= 0;
		end
		else if (counter == 14'b11000011010100)
			clklow <= 1;
		else
			counter <= counter + 1;
	end

endmodule
```

A.2.c Anti-rebote

```
`timescale 1ns / 1ps
module antideboucing(low_clk, btn, AD_btn);
	input low_clk, btn;
	output AD_btn;

	reg [15:0] SR_out = 0;

	always  @(posedge low_clk) begin
		SR_out = {SR_out[14:0],~btn};
	end

	assign AD_btn = &SR_out;

endmodule
```

A.1.i) Código fuente divisor de reloj por 2

```
`timescale 1ns / 1ps
module clkdiv2(clk, clkdiv);
	input clk;
	output reg clkdiv = 0;

	always  @(posedge clk) begin
		clkdiv = ~clkdiv;
	end

endmodule
```

A.2.d Memoria de variables de entrada

```
`timescale 1ns / 1ps
module mem_xs(index, clk, x);
	parameter size = 1024, filepath = "x1.txt";

	input [9:0] index;
	input clk;
```

```
	output reg [16:0] x = 0;

	reg [16:0] xmem[0:size-1];

	initial begin
		$readmemh(filepath, xmem);
	end

	always  @(posedge clk) begin

		x <= xmem[index];

	end

endmodule
```

A.2.e Red neuronal

```
`timescale 1ns / 1ps
module red_neuronal(x1, x2, start, mode, clk, ready, error, new_weight1, oldremainder1, new_weight2, oldremainder2,
d);

	input signed [16:0] x1;
	input signed [16:0] x2;

	input start;
	input mode; //Modo de red, 0 = sin entrenamiento, 1 = con actualizacion de pesos
	input clk;

	output reg ready = 0;		//inicializa ocupado cargando los valores iniciales a los pesos
	output reg signed [35:0] error;

	//referencia de red neuronal, valor deseado
	output reg signed [35:0] d;

	//temporales para resultados -- 2X
	output [15:0] new_weight1;
	output signed [17:0] oldremainder1;
	output [15:0] new_weight2;
	output signed [17:0] oldremainder2;

	reg [3:0] state = 0;
	wire signed [21:0] recorte_error = error[21:0];

	wire signed [8:0] constante = 9'd205;

	//Auxiliares -- 3X
	wire signed [10:0] x1nu = x1[16:6];
	wire signed [32:0] x1nu_error = x1nu*recorte_error;
	wire signed [16:0] recorte_delta1 = x1nu_error[31:15];
	wire signed [24:0] reescalada_delta1 = recorte_delta1*constante;
	wire signed [16:0] delta1_final = reescalada_delta1[23:7];

	wire signed [10:0] x2nu = x2[16:6];
	wire signed [32:0] x2nu_error = x2nu*recorte_error;
```

```
wire signed [16:0] recorte_delta2 = x2nu_error[31:15];
wire signed [24:0] reescalada_delta2 = recorte_delta2*constante;
wire signed [16:0] delta2_final = reescalada_delta2[23:7];

//Variables perceptron -- 4X
reg signed [17:0] dw1 = 0;
reg signed [17:0] dw2 = 0;

reg startmults = 0;
reg startmems = 0;

wire readymems, readymults;
wire signed [35:0] Y;

reg [6:0] mul_sinapsis_counter = 0;

//Multiplicadores de peso fijo para test perceptron -- 5X
//Variables multiplicador 1 de peso fijo
reg startm1 = 0;
reg readymu_1_up;
wire signed [17:0] wx1;
wire readym1;

//Variables multiplicador 2 de peso fijo
reg startm2 = 0;
reg readymu_2_up;
wire signed [17:0] wx2;
wire readym2;


//Creacion instancia perceptron y multiplicadores de peso fijo -- 6X
perceptron per1(x1, dw1, x2, dw2, clk, startmults, startmems, readymems, readymults, Y, new_weight1,
oldremainder1, new_weight2, oldremainder2);

multiplicador2 #(.nmult(44)) m1(startm1, clk, x1, 16'hF33B, wx1, readym1);
multiplicador2 #(.nmult(16)) m2(startm2, clk, x2, 16'h3B2C, wx2, readym2);

initial begin        //Inicializa las memorias para cargar los valores iniciales
        startmems <= 1;
end

always  @(posedge clk) begin // -- 7X
        if (~readym1) startm1 <= 0;
        if (~readym2) startm2 <= 0;

        if (~readymults) startmults <= 0;
        if (~readymems) startmems <= 0;

        case (state)
                0: begin
                        if (start) begin // -- 8X
                                //arranque multiplicadores de pesos fijos
                                startm1 <= 1;
                                readymu_1_up <= 0;

                                startm2 <= 1;
                                readymu_2_up <= 0;
```

```
                        mul_sinapsis_counter <= 2;
                        startmults <= 1;   //arranque de multiplicadores de perceptron
                        d <= 0;

                        error <= 1;
                        ready <= 0;
                        state <= state + 1;
                end
                else if (~startmems && readymems) begin
                        ready <= 1;
                end
        end
        1: begin // -- 9X
                if (readym1 && ~readymu_1_up) begin
                        mul_sinapsis_counter <= mul_sinapsis_counter - 1;
                        d <= d + wx1;
                        readymu_1_up <= 1;
                end
                else if (readym2 && ~readymu_2_up) begin
                        mul_sinapsis_counter <= mul_sinapsis_counter - 1;
                        d <= d + wx2;
                        readymu_2_up <= 1;
                end
                if (mul_sinapsis_counter == 0) state <= state + 1;
                // cuando contador llega a 0, se ha obtenido referencia
                //se sigue en siguiente paso

        end
        2: begin
                if (readymults) begin //si se han hecho las multiplicaciones, Y contiene W'*x
                        error <= d - Y;
                        state <= state + 1;
                end
        end
        3: begin //se calcula el delta y se deja este estado ocioso en espera del resultado
                state <= state + 1;
        end
        4: begin // -- 10X
                dw1 <= delta1_final;
                dw2 <= delta2_final;

                startmems <= 1;
                state <= state + 1;
        end
        5: begin
                if (readymems) begin
                        ready <= 1;
                        state <= 0;
                end
        end
        default: begin
                state <= 0;
        end

endcase
```

```
        end

endmodule
```

A.2.f Perceptrón

```
`timescale 1ns / 1ps
module perceptron(x1, dw1, x2, dw2, clk, startmults, startmems, readymems, readymults, Y, new_weight1,
oldremainder1, new_weight2, oldremainder2);

        input signed [16:0] x1;
        input signed [17:0] dw1;
        input signed [16:0] x2;
        input signed [17:0] dw2;

        input clk;
        input startmults, startmems;

        output readymems, readymults;
        output reg signed [35:0] Y;                                     //Salida perceptron

        //Salidas con fines de obtencion de resultados -- X
        output [15:0] new_weight1;
        output signed [17:0] oldremainder1;
        output [15:0] new_weight2;
        output signed [17:0] oldremainder2;

        reg [6:0] mem_sinapsis_counter = 0;
        reg [6:0] mul_sinapsis_counter = 0;

        reg [3:0] memstate = 0;
        reg [3:0] mulstate = 0;

        assign readymems = mem_sinapsis_counter == 0;
        assign readymults = mul_sinapsis_counter == 0;

        // Declaracion de variables -- X
        //-----------------------------------//
        //Variables sinapsis 44
        reg start_mult1 = 0;
        reg start_mem1 = 0;
        reg readyme_1_up;
        reg readymu_1_up;
        wire signed [17:0] y1;
        wire ready_mem1;
        wire ready_mult1;
        //-----------------------------------//
        //Variables sinapsis 16
        reg start_mult2 = 0;
        reg start_mem2 = 0;
        reg readyme_2_up;
        reg readymu_2_up;
        wire signed [17:0] y2;
        wire ready_mem2;
        wire ready_mult2;
```

```
	//Modulos sinapsis presentes -- X
	sinapsis #(.numero(44),.initial_index(223)) synapse1(dw1, x1, clk, start_mult1, start_mem1, y1, ready_mem1, ready_mult1, new_weight1, oldremainder1);
	sinapsis #(.numero(16),.initial_index(125)) synapse2(dw2, x2, clk, start_mult2, start_mem2, y2, ready_mem2, ready_mult2, new_weight2, oldremainder2);

	always  @(negedge clk) begin // -- X
		if (~ready_mem1) start_mem1 <= 0;
		if (~ready_mult1) start_mult1 <= 0;
		if (~ready_mem2) start_mem2 <= 0;
		if (~ready_mult2) start_mult2 <= 0;

		case (memstate)
			0: begin
				if (startmems) begin // -- X

					readyme_1_up <= 0;
					start_mem1 <= 1;

					readyme_2_up <= 0;
					start_mem2 <= 1;

					mem_sinapsis_counter <= 2;

					memstate <= memstate + 1;
				end
			end
			1: begin // -- X
				if (ready_mem1 && ~readyme_1_up) begin
					mem_sinapsis_counter <= mem_sinapsis_counter - 1;
					readyme_1_up <= 1;
				end
				else if (ready_mem2 && ~readyme_2_up) begin
					mem_sinapsis_counter <= mem_sinapsis_counter - 1;
					readyme_2_up <= 1;
				end
				if (mem_sinapsis_counter == 0) memstate <= 0;
			end
			default: begin
				memstate <= 0;
			end
		endcase

		case (mulstate)
			0: begin
				if (startmults) begin // -- X

					readymu_1_up <= 0;
					start_mult1 <= 1;

					readymu_2_up <= 0;
					start_mult2 <= 1;

					mul_sinapsis_counter <= 2;
					Y <= 0;
					mulstate <= mulstate + 1;
```

```
                        end
                    end
                    1: begin // -- X
                        if (ready_mult1 && ~readymu_1_up) begin
                            mul_sinapsis_counter <= mul_sinapsis_counter - 1;
                            Y <= Y + y1;
                            readymu_1_up <= 1;
                        end
                        else if (ready_mult2 && ~readymu_2_up) begin
                            mul_sinapsis_counter <= mul_sinapsis_counter - 1;
                            Y <= Y + y2;
                            readymu_2_up <= 1;
                        end

                        if (mul_sinapsis_counter == 0) mulstate <= 0;
                    end
                    default: begin
                        mulstate <= 0;
                    end
                endcase
        end
endmodule
```

A.2.g Sinapsis

```
`timescale 1ns / 1ps
module sinapsis(deltaw, x, clk, start_mult, start_mem, y, ready_mem, ready_mult, new_weight, oldremainder);
        parameter numero = 5, initial_index = 15;

        input signed [17:0] deltaw;
        input signed [16:0] x;
        input clk;
        input start_mult;
        input start_mem;

        output signed [17:0] y;
        output ready_mem;
        output ready_mult;
        output [15:0] new_weight;//salida para efectos de visualizacion de resultados

        //Señales memoria
        reg signed [9:0] oldindex;  //valor inicial se carga como parametro para ser aleatorio
        output reg signed [17:0] oldremainder;

        wire signed [17:0] newremainder;
        wire [12:0] newindex;
        wire ready_mem;

        //Modulos memoria y multiplicador
        memoria #(.memory(numero)) memory(start_mem, clk, oldindex, deltaw, oldremainder, newremainder,
newindex, new_weight, ready_mem);
        multiplicador #(.nmult(numero)) multtester(start_mult, clk, x, new_weight, y, ready_mult);

        initial begin       //carga de indice de peso inicial en memoria
                oldindex <= initial_index;
```

```
                oldremainder <= 0;
        end

        //Cada vez que la memoria completa su operacion, se asignan los nuevos valores como antiguos para
        //la siguiente actualizacion
        always  @(posedge ready_mem) begin
                oldindex <= newindex;
                oldremainder <= newremainder;
        end

endmodule
```

A.2.h Memoria

```
`timescale 1ns / 1ps
module memoria(start, clk, oldindex, change, oldremainder, newremainder, newindex, new_weight, ready);
        parameter memory = 5;

        input start, clk;
        input signed [9:0] oldindex;
        input signed [17:0] change;
        input signed [17:0] oldremainder;

        output reg signed [17:0] newremainder;
        output reg [12:0] newindex;
        output reg [15:0] new_weight;
        output reg ready = 1;

        wire signed [17:0] parte_entera;
        wire [4:0] v,c;
        reg [3:0] state = 0;

        reg signed [13:0] signed_newindex;
        reg signed [17:0] realchange;

        reg start_div = 0;
        reg signed [21:0] peso_aprox;
        wire signed [17:0] extraccion_peso = peso_aprox[17:0];

        reg [8:0] index;
        wire [16:0] memOut;
        reg [12:0] levels;
        reg [12:0] pulsos;
        reg signed [17:0] delta;
        reg signo;
        wire signed [17:0] result;
        wire ready_div;

        wire [12:0] jump = memOut[12:0];        //jump es la cantidad de niveles (salto en peso y parametros)
        wire [15:0] wmem = memOut[15:0];        //extraccion de 16 bits LSB
        wire [6:0] nmem = memOut[6:0];          //extraccion de 7 bits LSB
        wire signed [11:0] delta_short = delta[11:0];

        divider #(.decimal_bits(0)) divisor(realchange, delta, clk, start_div, parte_entera, ready_div);
```

```
//Bloque generacion de memoria mediante codigo automatizado de matlab --- X
generate
        case (memory)
                1: memmem #(.size(48), .filepath("data_mem1.txt")) tablamemW(index, clk, memOut);
                2: memmem #(.size(43), .filepath("data_mem2.txt")) tablamemW(index, clk, memOut);
                3: memmem #(.size(45), .filepath("data_mem3.txt")) tablamemW(index, clk, memOut);
                4: memmem #(.size(98), .filepath("data_mem4.txt")) tablamemW(index, clk, memOut);
                5: memmem #(.size(58), .filepath("data_mem5.txt")) tablamemW(index, clk, memOut);
                6: memmem #(.size(76), .filepath("data_mem6.txt")) tablamemW(index, clk, memOut);
                7: memmem #(.size(79), .filepath("data_mem7.txt")) tablamemW(index, clk, memOut);
                8: memmem #(.size(46), .filepath("data_mem8.txt")) tablamemW(index, clk, memOut);
                9: memmem #(.size(42), .filepath("data_mem9.txt")) tablamemW(index, clk, memOut);
                10: memmem #(.size(45), .filepath("data_mem10.txt")) tablamemW(index, clk,memOut);
                11: memmem #(.size(51), .filepath("data_mem11.txt")) tablamemW(index, clk,memOut);
                12: memmem #(.size(45), .filepath("data_mem12.txt")) tablamemW(index, clk,memOut);
                13: memmem #(.size(42), .filepath("data_mem13.txt")) tablamemW(index, clk,memOut);
                14: memmem #(.size(44), .filepath("data_mem14.txt")) tablamemW(index, clk,memOut);
                15: memmem #(.size(40), .filepath("data_mem15.txt")) tablamemW(index, clk,memOut);
                16: memmem #(.size(258), .filepath("data_mem16.txt")) tablamemW(index, clk,memOut);
                17: memmem #(.size(62), .filepath("data_mem17.txt")) tablamemW(index, clk,memOut);
                18: memmem #(.size(80), .filepath("data_mem18.txt")) tablamemW(index, clk,memOut);
                19: memmem #(.size(43), .filepath("data_mem19.txt")) tablamemW(index, clk,memOut);
                20: memmem #(.size(50), .filepath("data_mem20.txt")) tablamemW(index, clk,memOut);
                21: memmem #(.size(47), .filepath("data_mem21.txt")) tablamemW(index, clk,memOut);
                22: memmem #(.size(61), .filepath("data_mem22.txt")) tablamemW(index, clk,memOut);
                23: memmem #(.size(45), .filepath("data_mem23.txt")) tablamemW(index, clk,memOut);
                24: memmem #(.size(39), .filepath("data_mem24.txt")) tablamemW(index, clk,memOut);
                25: memmem #(.size(44), .filepath("data_mem25.txt")) tablamemW(index, clk,memOut);
                26: memmem #(.size(103), .filepath("data_mem26.txt")) tablamemW(index, clk,memOut);
                27: memmem #(.size(72), .filepath("data_mem27.txt")) tablamemW(index, clk,memOut);
                28: memmem #(.size(45), .filepath("data_mem28.txt")) tablamemW(index, clk,memOut);
                29: memmem #(.size(48), .filepath("data_mem29.txt")) tablamemW(index, clk,memOut);
                30: memmem #(.size(57), .filepath("data_mem30.txt")) tablamemW(index, clk,memOut);
                31: memmem #(.size(41), .filepath("data_mem31.txt")) tablamemW(index, clk,memOut);
                32: memmem #(.size(81), .filepath("data_mem32.txt")) tablamemW(index, clk,memOut);
                33: memmem #(.size(49), .filepath("data_mem33.txt")) tablamemW(index, clk,memOut);
                34: memmem #(.size(55), .filepath("data_mem34.txt")) tablamemW(index, clk,memOut);
                35: memmem #(.size(52), .filepath("data_mem35.txt")) tablamemW(index, clk,memOut);
                36: memmem #(.size(127), .filepath("data_mem36.txt")) tablamemW(index, clk,memOut);
                37: memmem #(.size(59), .filepath("data_mem37.txt")) tablamemW(index, clk,memOut);
                38: memmem #(.size(108), .filepath("data_mem38.txt")) tablamemW(index, clk,memOut);
                39: memmem #(.size(44), .filepath("data_mem39.txt")) tablamemW(index, clk,memOut);
                40: memmem #(.size(50), .filepath("data_mem40.txt")) tablamemW(index, clk,memOut);
                41: memmem #(.size(42), .filepath("data_mem41.txt")) tablamemW(index, clk,memOut);
                42: memmem #(.size(56), .filepath("data_mem42.txt")) tablamemW(index, clk,memOut);
                43: memmem #(.size(43), .filepath("data_mem43.txt")) tablamemW(index, clk,memOut);
                44: memmem #(.size(369), .filepath("data_mem44.txt")) tablamemW(index, clk,memOut);
                45: memmem #(.size(68), .filepath("data_mem45.txt")) tablamemW(index, clk,memOut);
                46: memmem #(.size(65), .filepath("data_mem46.txt")) tablamemW(index, clk,memOut);
                47: memmem #(.size(39), .filepath("data_mem47.txt")) tablamemW(index, clk,memOut);
                48: memmem #(.size(135), .filepath("data_mem48.txt")) tablamemW(index, clk,memOut);
                49: memmem #(.size(43), .filepath("data_mem49.txt")) tablamemW(index, clk,memOut);
                50: memmem #(.size(50), .filepath("data_mem50.txt")) tablamemW(index, clk,memOut);
                51: memmem #(.size(236), .filepath("data_mem51.txt")) tablamemW(index, clk,memOut);
                52: memmem #(.size(158), .filepath("data_mem52.txt")) tablamemW(index, clk,memOut);
                53: memmem #(.size(85), .filepath("data_mem53.txt")) tablamemW(index, clk,memOut);
                54: memmem #(.size(264), .filepath("data_mem54.txt")) tablamemW(index, clk,memOut);
```

```
                55: memmem #(.size(81), .filepath("data_mem55.txt")) tablamemW(index, clk,memOut);
                56: memmem #(.size(64), .filepath("data_mem56.txt")) tablamemW(index, clk,memOut);
                57: memmem #(.size(55), .filepath("data_mem57.txt")) tablamemW(index, clk,memOut);
               58: memmem #(.size(191), .filepath("data_mem58.txt")) tablamemW(index, clk,memOut);
                59: memmem #(.size(43), .filepath("data_mem59.txt")) tablamemW(index, clk,memOut);
                60: memmem #(.size(46), .filepath("data_mem60.txt")) tablamemW(index, clk,memOut);
               61: memmem #(.size(246), .filepath("data_mem61.txt")) tablamemW(index, clk,memOut);
                62: memmem #(.size(47), .filepath("data_mem62.txt")) tablamemW(index, clk,memOut);
                63: memmem #(.size(54), .filepath("data_mem63.txt")) tablamemW(index, clk,memOut);
                64: memmem #(.size(46), .filepath("data_mem64.txt")) tablamemW(index, clk,memOut);
        endcase
endgenerate
//Termino generacion de codigo automatizado

//Borrar bloque generate y copiar memoria.v como memoria.v.backup en carpeta de emulador en matlab
//si se introducen cambios y validarlos en archivo de respaldo

always  @(posedge clk) begin
if (~ready_div) start_div <= 0;

        case (state)
                0: begin
                devstate <= 0;
                        if (start) begin                //se espera señal se inicio de conversion
                                ready <= 0;             //se resetea flag ready
                                index <= 0;             //puntero vuelve a 0, apunta a numero de niveles
                                pulsos <= 0;            //se resetea contador de pulsos
                                start_div <= 0;         //gatillo inicio division
                                state <= state + 1;//se inicia proceso de actualizacion...
                        end
                end
                1: begin
                        levels <= memOut[12:0];  //se almacena cantidad de niveles
                        index <= index + memOut[12:0] + 1; //se posiciona puntero en delta promedio
                        realchange <= change + oldremainder; //Se considera remanente anterior

                        state <= state + 1;             //siguiente estado
                end
                2: begin //se obtiene delta a utilizar
                        delta <= {1'b0,memOut[15:0]};       //se extrae delta promedio

                        if (realchange > 0) begin   //si la actualizacion es positiva
                                signo <= 1;             //se activa flag que asi lo indica
                        end
                        else begin
                                signo <= 0;             //se resetea flag si signo de actualizacion es negativo
                        end

                        start_div <= 1;                 //inicio division para obtencio de cantidad de pulsos

                        state <= state + 1;             //siguiente estado
                end
                3: begin
                        if (ready_div) begin
                                peso_aprox <= parte_entera*delta_short;
                                signed_newindex <= oldindex + parte_entera;
```

```
                                        state <= state + 1;
                                end
                        end
                        4: begin
                                if (signed_newindex < 1) begin      //si indice es menor que el minimo (1)
                                        newindex <= 1;
                                        index <= 1; //se apunta a posicion de memoria para obtener el peso
                                        newremainder <= 0; //se resetea remanente por llegar a un limite
                                end //si indice supera cantidad de niveles
                                else if (signed_newindex > {1'b0,levels} ) begin
                                        newindex <= levels;
                                        index <= levels;
                                        newremainder <= 0;
                                        //se resetea remanente por llegar a un limite
                                end
                                else begin
                                        //si indice se encuentra en rango aceptable
                                        newindex <= signed_newindex[12:0];
                                        index <= signed_newindex[12:0];
                                        newremainder <= realchange - extraccion_peso;
                                end

                                state <= state + 1;
                        end
                        5: begin
                                new_weight <= memOut[15:0];

                                ready <= 1;
                                state <= 0; //retorno a estado inicial
                        end
                        default: begin
                                state <= 0;
                        end
                endcase
        end

endmodule
```

A.2.i Divisor

```
`timescale 1ns / 1ps
module divider(dividendo, divisor, clk, start, cuociente, ready);
        parameter decimal_bits = 8;

        input signed [17:0] dividendo, divisor;
        input clk, start;

        output signed [17:0] cuociente;
        output reg ready = 1;

        reg [4:0] v;
        reg [2:0] state = 0;
        reg [4:0] c;

        reg [17:0] divisor_unsig;
```

```
wire [17:0] divisor_double = {divisor_unsig[16:0], 1'b0};

reg signed [17:0] cuociente_unsig;

reg quotient;

reg [17:0] p_entera, p_decimal;

wire dividendo_sign = dividendo[17];
wire divisor_sign = divisor[17];
wire cuociente_sign = dividendo_sign ^ divisor_sign;

assign cuociente = cuociente_sign ? -cuociente_unsig : cuociente_unsig;

always  @ (negedge clk) begin
        case (state)
                0: begin
                        if (start) begin
                                if ( dividendo[17] == 1 ) begin         //si dividendo es negativo
                                        p_entera <= -dividendo;         //se deja positivo
                                end
                                else begin
                                        p_entera <= dividendo;
                                end

                                if ( divisor[17] == 1 )                 //si divisor es negativo
                                        divisor_unsig <= (-divisor);    //se deja positivo
                                else
                                        divisor_unsig <= divisor;

                                cuociente_unsig <= 0;
                                quotient <= 0;
                                c <= 0;
                                v <= 0;

                                p_decimal <= 0;

                                ready <= 0;
                                state <= state + 1;
                        end
                end
                1: begin        //Etapa de Prenormalizacion (Dividendo < 2 * Divisor)
                        if ( p_entera >= divisor_double ) begin

                                if (p_entera[0] == 1) p_decimal <= {1'b1,p_decimal[17:1]};
                                else p_decimal <= {1'b0, p_decimal[17:1]};

                                state <= 5;
                                c <= c + 1;
                        end
                        else begin
                                state <= state + 1;
                        end
                end
                2: begin        //Etapa de generacion de cuociente (1 de 2)
                        if( p_entera >= divisor_unsig ) begin
                                p_entera <= p_entera - divisor_unsig;
```

```
                                        quotient <= 1;
                                end
                                else quotient <= 0;

                                state <= state + 1;
                        end
                        3: begin            //Etapa de generacion de cuociente (2 de 2) y Postnormalizacion
                                if (p_decimal[17] == 1) p_entera <= {p_entera[16:0],1'b1};
                                else p_entera <= p_entera << 1;

                                p_decimal <= p_decimal << 1;

                                if( v != (decimal_bits + 1 + c) ) begin
                                        cuociente_unsig <= {cuociente_unsig[16:0] , quotient};
                                        state <= state - 1;
                                end
                                else begin
                                        state <= state + 1;
                                end

                                v <= v + 1;
                        end
                        4: begin
                                ready <= 1;
                                state <= 0;
                        end
                        5: begin
                                p_entera <= {1'b0, p_entera[17:1]};
                                state <= 1;
                        end

                        default: begin
                                state <= 0;
                        end
                endcase
        end

endmodule
```

A.2.j Memoria de pesos

```
`timescale 1ns / 1ps
module memmem(index, clk, memOut);
        parameter size = 50, filepath = "data_mem20.txt";

        input [8:0] index;
        input clk;

        output reg [16:0] memOut = 0;
        reg [16:0] pesosmem[0:size-1];      //Iniciacion de pesos de memoria de tamaño dado por size

        initial begin
                $readmemh(filepath, pesosmem);
        end
```

```
        always  @(negedge clk) begin
                        memOut <= pesosmem[index];
        end

        //Organizacion Memoria

        //#niveles
        //peso(1)
        //peso(2)
        // *
        // *
        // *
        //peso(niveles)
        //delta promedio

Endmodule
```

A.2.k Multiplicador

```
`timescale 1ns / 1ps
module multiplicador(start, clk, x, w, product, ready);
        parameter nmult = 5;

        input clk, start;
        input signed [16:0] x;              //multiplicando de 16 bits mas bit de signo
        input [15:0] w;                     //multiplicando de peso, 16 bits sin signo

        output signed [17:0] product;       //resultado de la multiplicacion
        output ready;                       //flag que indica cuando la conversion ha terminado

        reg signed [33:0] y;          //salida de tanh de ec. de la recta
        wire signed [16:0] y_short = y[32:16]; //version acortada de tanh

        reg signed [33:0] product_long = 0;

        assign product = product_long[32:15];

        reg [10:0] index = 0;               //indice de puntero de memoria de parametros y pesos
        wire signed [16:0] abc_memOut;      //salida de memoria de parametros y pesos

        reg signed [16:0] a;                        //registro con parametro a
        reg signed [14:0] b;                        //registro con parametro b
        wire signed [14:0] bwire = abc_memOut[14:0];
        reg signed [33:0] c;                        //registro con parametro c
        reg signed [31:0] bx;               //resultado de multiplicacion de b y x
        wire signed [16:0] bx2 = bx[30:14]; //version achicada de bx para calcular salida tanh
        wire signed [16:0] bx_positive;

        reg signed [16:0] dist;             //medicion de distancia en busqueda de peso
        reg [16:0] dist_winner;             //indice de menor distancia
        reg [12:0] mult;

        wire signed [17:0] constante2 = 18'h12C70;  //constante para dejar en equivalencia de 34 bits
        wire signed [17:0] constante3 = 18'h12C6A; //constante para dejar en equivalencia de 34 bits
```

```
wire signo = bx2[16];
wire [8:0] indextanh2 = signo ?  ~bx[29:21] + 1 : bx[29:21];    //indice de 9 bits para memoria de tanh´s

reg [3:0] state = 0;
reg signed [16:0] m;
reg signed [33:0] offset;
reg [9:0] barridos;
reg [9:0] w_counter;
wire [15:0] wmem = abc_memOut[15:0];     //extraccion de 16 bits LSB
wire signed [16:0] Out_m;                //salidas m y b de memoria tanh
wire signed [14:0] Out_b;                //salidas m y b de memoria tanh

assign bx_positive =  bx2[16] ? -bx2 : bx2;

//memoria de tanh con parametros m y b de recta y = mx + b, modo free running
mem_m tanh_m_b(indextanh2, clk, Out_m, Out_b);

//Creacion de generate a partir de esta linea --- X
generate
        case (nmult)
        1: memabc #(.size(185), .filepath("data_mult1.txt")) memWabc(index, clk, abc_memOut);
        2: memabc #(.size(165), .filepath("data_mult2.txt")) memWabc(index, clk, abc_memOut);
        3: memabc #(.size(173), .filepath("data_mult3.txt")) memWabc(index, clk, abc_memOut);
        4: memabc #(.size(385), .filepath("data_mult4.txt")) memWabc(index, clk, abc_memOut);
        5: memabc #(.size(225), .filepath("data_mult5.txt")) memWabc(index, clk, abc_memOut);
        6: memabc #(.size(297), .filepath("data_mult6.txt")) memWabc(index, clk, abc_memOut);
        7: memabc #(.size(309), .filepath("data_mult7.txt")) memWabc(index, clk, abc_memOut);
        8: memabc #(.size(177), .filepath("data_mult8.txt")) memWabc(index, clk, abc_memOut);
        9: memabc #(.size(161), .filepath("data_mult9.txt")) memWabc(index, clk, abc_memOut);
        10: memabc #(.size(173), .filepath("data_mult10.txt")) memWabc(index, clk, abc_memOut);
        11: memabc #(.size(197), .filepath("data_mult11.txt")) memWabc(index, clk, abc_memOut);
        12: memabc #(.size(173), .filepath("data_mult12.txt")) memWabc(index, clk, abc_memOut);
        13: memabc #(.size(161), .filepath("data_mult13.txt")) memWabc(index, clk, abc_memOut);
        14: memabc #(.size(169), .filepath("data_mult14.txt")) memWabc(index, clk, abc_memOut);
        15: memabc #(.size(153), .filepath("data_mult15.txt")) memWabc(index, clk, abc_memOut);
        16: memabc #(.size(1025), .filepath("data_mult16.txt")) memWabc(index, clk, abc_memOut);
        17: memabc #(.size(241), .filepath("data_mult17.txt")) memWabc(index, clk, abc_memOut);
        18: memabc #(.size(313), .filepath("data_mult18.txt")) memWabc(index, clk, abc_memOut);
        19: memabc #(.size(165), .filepath("data_mult19.txt")) memWabc(index, clk, abc_memOut);
        20: memabc #(.size(193), .filepath("data_mult20.txt")) memWabc(index, clk, abc_memOut);
        21: memabc #(.size(181), .filepath("data_mult21.txt")) memWabc(index, clk, abc_memOut);
        22: memabc #(.size(237), .filepath("data_mult22.txt")) memWabc(index, clk, abc_memOut);
        23: memabc #(.size(173), .filepath("data_mult23.txt")) memWabc(index, clk, abc_memOut);
        24: memabc #(.size(149), .filepath("data_mult24.txt")) memWabc(index, clk, abc_memOut);
        25: memabc #(.size(169), .filepath("data_mult25.txt")) memWabc(index, clk, abc_memOut);
        26: memabc #(.size(405), .filepath("data_mult26.txt")) memWabc(index, clk, abc_memOut);
        27: memabc #(.size(281), .filepath("data_mult27.txt")) memWabc(index, clk, abc_memOut);
        28: memabc #(.size(173), .filepath("data_mult28.txt")) memWabc(index, clk, abc_memOut);
        29: memabc #(.size(185), .filepath("data_mult29.txt")) memWabc(index, clk, abc_memOut);
        30: memabc #(.size(221), .filepath("data_mult30.txt")) memWabc(index, clk, abc_memOut);
        31: memabc #(.size(157), .filepath("data_mult31.txt")) memWabc(index, clk, abc_memOut);
        32: memabc #(.size(317), .filepath("data_mult32.txt")) memWabc(index, clk, abc_memOut);
        33: memabc #(.size(189), .filepath("data_mult33.txt")) memWabc(index, clk, abc_memOut);
        34: memabc #(.size(213), .filepath("data_mult34.txt")) memWabc(index, clk, abc_memOut);
        35: memabc #(.size(201), .filepath("data_mult35.txt")) memWabc(index, clk, abc_memOut);
        36: memabc #(.size(501), .filepath("data_mult36.txt")) memWabc(index, clk, abc_memOut);
        37: memabc #(.size(229), .filepath("data_mult37.txt")) memWabc(index, clk, abc_memOut);
```

```
            38: memabc #(.size(425), .filepath("data_mult38.txt")) memWabc(index, clk, abc_memOut);
            39: memabc #(.size(169), .filepath("data_mult39.txt")) memWabc(index, clk, abc_memOut);
            40: memabc #(.size(193), .filepath("data_mult40.txt")) memWabc(index, clk, abc_memOut);
            41: memabc #(.size(161), .filepath("data_mult41.txt")) memWabc(index, clk, abc_memOut);
            42: memabc #(.size(217), .filepath("data_mult42.txt")) memWabc(index, clk, abc_memOut);
            43: memabc #(.size(165), .filepath("data_mult43.txt")) memWabc(index, clk, abc_memOut);
            44: memabc #(.size(1469), .filepath("data_mult44.txt")) memWabc(index, clk, abc_memOut);
            45: memabc #(.size(265), .filepath("data_mult45.txt")) memWabc(index, clk, abc_memOut);
            46: memabc #(.size(253), .filepath("data_mult46.txt")) memWabc(index, clk, abc_memOut);
            47: memabc #(.size(149), .filepath("data_mult47.txt")) memWabc(index, clk, abc_memOut);
            48: memabc #(.size(533), .filepath("data_mult48.txt")) memWabc(index, clk, abc_memOut);
            49: memabc #(.size(165), .filepath("data_mult49.txt")) memWabc(index, clk, abc_memOut);
            50: memabc #(.size(193), .filepath("data_mult50.txt")) memWabc(index, clk, abc_memOut);
            51: memabc #(.size(937), .filepath("data_mult51.txt")) memWabc(index, clk, abc_memOut);
            52: memabc #(.size(625), .filepath("data_mult52.txt")) memWabc(index, clk, abc_memOut);
            53: memabc #(.size(333), .filepath("data_mult53.txt")) memWabc(index, clk, abc_memOut);
            54: memabc #(.size(1049), .filepath("data_mult54.txt")) memWabc(index, clk, abc_memOut);
            55: memabc #(.size(317), .filepath("data_mult55.txt")) memWabc(index, clk, abc_memOut);
            56: memabc #(.size(249), .filepath("data_mult56.txt")) memWabc(index, clk, abc_memOut);
            57: memabc #(.size(213), .filepath("data_mult57.txt")) memWabc(index, clk, abc_memOut);
            58: memabc #(.size(757), .filepath("data_mult58.txt")) memWabc(index, clk, abc_memOut);
            59: memabc #(.size(165), .filepath("data_mult59.txt")) memWabc(index, clk, abc_memOut);
            60: memabc #(.size(177), .filepath("data_mult60.txt")) memWabc(index, clk, abc_memOut);
            61: memabc #(.size(977), .filepath("data_mult61.txt")) memWabc(index, clk, abc_memOut);
            62: memabc #(.size(181), .filepath("data_mult62.txt")) memWabc(index, clk, abc_memOut);
            63: memabc #(.size(209), .filepath("data_mult63.txt")) memWabc(index, clk, abc_memOut);
            64: memabc #(.size(177), .filepath("data_mult64.txt")) memWabc(index, clk, abc_memOut);
            endcase
endgenerate
//Termino generacion de codigo automatizado

assign ready = (state == 0) && !start;

always  @(posedge clk) begin
            case (state)
                        0: begin
                                    if (start) begin      //se espera señal se inicio de conversion
                                                dist_winner <= 16'hFFFF; //se resetea valor de distancia
                                                index <= 0;                 //puntero de memoria se vuelve a 0
                                                w_counter <= 1;
                                                m <= 0;                     //reseteo de pendiente de recta de tanh
                                                offset <= 0;                //reseteo de offset de recta de tanh
                                                product_long <= 0;          //se resetea resultado del producto

                                                state <= state + 1;//se inicia proceso de busqueda de peso en memoria
                                    end
                        end
                        1: begin //memoria retorna cantidad de barridos en medio pulso anterior
                                    index <= index + 1; //incremento de puntero para obtencion de primer peso

                                    barridos <= abc_memOut[9:0];
                                    state <= state + 1; //se sigue en estado 2
                        end
                        2: begin //con peso extraido de memoria, se interpola con nearest neighborhood
                                    if (w > wmem) begin         //distancia entre pesos (memoria y peso ingresado)
                                                dist <= w - wmem;
                                    end
```

```
                else begin//se evalua que peso es mayor para dejar positivo el valor de distancia
                        dist <= wmem - w;
                end

                state <= state + 1;

        end
        3: begin
                if (dist < dist_winner) begin //si distancia se achica se sigue iterando
                        dist_winner <= dist;                            //se asigna nueva distancia
                        w_counter <= w_counter + 1; //se incrementa contador de pesos

                        //si se ha sobrepasado la cantidad de pesos --> se ha encontrado el peso
                        if ( (w_counter + 1) > barridos) begin
                                index <= index + 1;             //se posiciona en constante a
                                state <= state + 1;             //siguiente estado
                        end
                        else begin
                                index <= index + 4; //siguiente peso, no se ha llegado al final
                                state <= 2;
                        end
                end
                else begin //si la distancia aumenta es por que se ha pasado el peso mas cercano
                        index <= index - 3;             //se posiciona en constante a
                        state <= state + 1;             //se siguen extrayendo parametros a, b y c
                end
        end
        4: begin
                index <= index + 1;                     //se posiciona en constante b
                a <= abc_memOut[16:0];  //extraccion de a
                state <= state + 1;
        end
        5: begin
                index <= index + 1;             //se posiciona en constante c
                b <= abc_memOut[14:0];  //extraccion de b y calculo de desfase,

                //se calcula desfase de tanh, se modifica tambien indice de memoria tanh
                bx <= bwire*x;

                state <= state + 1;
        end
        6: begin
                c <= abc_memOut*constante2;       //extraccion de c y amplificacion a 34 bits

                state <= state + 1;
        end
        7: begin //se tienen en Out_m y Out_b, m y b de ec de la recta y = m*(b*x) + b
                m <= Out_m;
                offset <= Out_b*constante3;

                state <= state + 1;
        end
        8: begin
                y <= m*bx_positive + offset; //calculo de tanh por rectas

                state <= state + 1;
        end
```

```
                    9: begin
                        if (signo) begin
                            y <= -y; //si signo de b*x es negativo, entonces se invierte de signo
                        end

                        state <= state + 1;
                    end
                    10: begin//tanh contiene dato extraido de memoria tangente hiperbolica mediante puntero
                        product_long <= a*y_short + c;      //se calcula la curva final
                        state <= state + 1;                 //se vuelve a estado inicial en espera
                    end
                    11: begin
                        state <= 0;
                    end

                    default: begin
                        state <= 0;
                    end
                endcase
        end
endmodule
```

A.2.l Memoria de parámetros de ecuación de la recta

```
`timescale 1ns / 1ps
module mem_m(index, clk, Out_m, Out_b);
        input [8:0] index;
        input clk;

        output reg [16:0] Out_m = 0;
        output reg [14:0] Out_b = 0;
        reg [16:0] mem_m[0:511]; //Memoria de pendientes de tanh
        reg [14:0] mem_b[0:511]; //Memoria de offsets de tanh

        initial begin
                $readmemh("tanh_b.txt", mem_b);
                $readmemh("tanh_m.txt", mem_m);
        end

        always  @(negedge clk) begin
                Out_m <= mem_m[index];
                Out_b <= mem_b[index];
        end

endmodule
```

A.2.m Memoria de parametros *a*, *b* y *c*

```
`timescale 1ns / 1ps
module memabc(index, clk, memOut);
        parameter size = 225, filepath = "data_mult5.txt";

        input [10:0] index;
        input clk;
```

```
        output reg [16:0] memOut = 0;

        reg [16:0] abcmem[0:size-1];

        initial begin
                $readmemh(filepath, abcmem);
        end

        always  @(negedge clk) begin
                memOut <= abcmem[index];
        end

        //Organizacion Memoria w abc

        //#sweeps
        //peso a b c    //esto se repite sweeps veces

endmodule
```

A.2.n Banco de prueba red neuronal

```
`timescale 1ns / 1ps
module red_neuronal_testbench();

        // Memorias para presentacion a la red -- 1X
        reg signed [16:0] xx1[0:1023];
        reg signed [16:0] x1;
        wire [15:0] new_weight1;
        integer handle1;
        wire signed [17:0] oldremainder1;

        reg signed [16:0] xx2[0:1023];
        reg signed [16:0] x2;
        wire [15:0] new_weight2;
        integer handle2;
        wire signed [17:0] oldremainder2;

        integer counter = 0;

        reg start = 0;
        reg mode; //Modo de red, 0 = sin entrenamiento, 1 = con actualizacion de pesos
        reg clk = 0;

        wire signed [35:0] d;
        wire ready;
        wire [35:0] error;

        // declaracion de red neuronal -- 2X
        red_neuronal test(x1, x2, start, mode, clk, ready, error, new_weight1, oldremainder1, new_weight2,
oldremainder2, d);

        initial begin
                $readmemh("x1.txt", xx1);
                handle1 = $fopen("peso1.txt");
```

```
		$readmemh("x2.txt", xx2);
		handle2 = $fopen("peso2.txt");
	end

	always  @(posedge ready) begin
		$fdisplay(handle1,"%d",new_weight1);
		$fdisplay(handle2,"%d",new_weight2);
	end

	always  @(negedge clk) begin
		if (ready) begin // -- 3X
			if (counter != 1023) begin
				x1 <= xx1[counter];
				x2 <= xx2[counter];
				start <= 1;
				counter <= counter + 1;
			end
			else begin
				$finish;
			end
		end
		else begin
			start <= 0;
		end

	end

	always  #5 clk = ~clk;

endmodule
```

A.2.o Banco de prueba multiplicador

```
`timescale 1ns / 1ps
module test_mult();
	parameter multiplier = 5;

	reg start;
	reg clk = 0;
	reg signed [16:0] x;
	reg [15:0] w;

	wire signed [17:0] product;
	wire ready;

	integer handle1, handle2, handle3;
	integer i=0;
	integer pesocounter = 1;
	integer puntos = 101;
	integer xcounter = 102;

	integer param_copy = multiplier;

	multiplicador #(multiplier) multtester(start, clk, x, w, product, ready);
```

```
initial begin
	handle1 = $fopen("curva.txt");
	$fdisplay(handle1,"%d",param_copy);
end

always  @(posedge ready) begin
	i = i + 1;
	$fdisplay(handle1,"%d",product);
end

always  @(negedge clk) begin
	if (ready) begin
		if (pesocounter <= 8 ) begin
			if (xcounter <= puntos) begin
				x = x + 1310;
				xcounter = xcounter + 1;
			end
			else begin //si se llego a limite
				x = -65535;
				if (pesocounter == 8) begin
					pesocounter = 9;
					$fclose(handle1);
				end
				xcounter = 2;
			end
			start = 1;
		end
		else begin//significa que ya termino
			$finish;
		end
	end
	else begin		//si ready cae
		start = 0;
	end

end

always  @(x) begin
	if (x == -65535) begin
		case (pesocounter)
			1: begin
				w = 16'h0;
				pesocounter = pesocounter + 1;
			end
			2: begin
				w = w + 10922;
				pesocounter = pesocounter + 1;
			end
			3: begin
				w = w + 10922;
				pesocounter = pesocounter + 1;
			end
			4: begin
				w = w + 10922;
				pesocounter = pesocounter + 1;
			end
			5: begin
```

```
                                        w = w + 10922;
                                        pesocounter = pesocounter + 1;
                                end
                                6: begin
                                        w = w + 10922;
                                        pesocounter = pesocounter + 1;
                                end
                                7: begin
                                        w = w + 10922;
                                        pesocounter = pesocounter + 1; //pesocounter = 8;
                                end
                                default: begin
                                        pesocounter = 9;
                                end
                        endcase
                end
        end

        always  #5 clk = ~clk;

endmodule
```

A.2.p Herramientas - driver LCD

```
`timescale 1ns / 1ps
module LCD_Driver(ASCII_msg, number2, mode, start, clk, ready, TxD_LCD);

        input [16*8-1:0] ASCII_msg;
        input signed [31:0] number2;
        input [3:0] mode;
        input start;
        input clk;

        output TxD_LCD;
        output reg ready = 1;


        reg signed [31:0] number;
        //bits modo
        wire mode_A = mode[0];  //si mode_A es ON --> Se limpia pantalla antes de ingresar nuevo dato
        wire mode_B = mode[1];  //si mode_B es ON --> Se envia a pantalla mensaje + un numero
        wire mode_C = mode[2];  //si mode_C es ON --> Se envia numero respetando su signo
        wire mode_D = mode[3];  //futuras opciones

        reg [3:0] state = 0;
        reg [3:0] nextstate;

        //variables displaymsg
        reg show = 0;
        reg [16*8-1:0] msg_buffer = 0;
        wire LCD_ready;

        //variables number2ASCII
        reg start_conversion = 0;
        wire conversion_ready;
```

```
wire [11*8-1:0] ACSIInumber; //10 Caracteres maximo para conversion

wire signo = number[31];
wire signed [31:0] numberLCD = (signo && mode_C) ? -number : number;

displaymsg      showmsg(msg_buffer, clk, show, LCD_ready, TxD_LCD);
converter2      num2ascii(numberLCD, clk, start_conversion, conversion_ready, ACSIInumber);

always  @(posedge clk) begin
        if (~conversion_ready) start_conversion <= 0;
        if (~LCD_ready) show <= 0;

        case (state)
                0: begin
                        if (start) begin
                                ready <= 0;
                                start_conversion <= 1;//se convierte inmediatamente numero a ASCII
                                number <= number2;

                                if (mode_A) begin       //Comando para limpiar pantalla
                                        msg_buffer <= 24'h1B5B6A;
                                        show <= 1;      //Envia a LCD
                                        nextstate <= 2;
                                        state <= state + 1;
                                end
                                else begin
                                        state <= state + 2;//no se limpia, se envia directamente a LCD
                                end
                        end
                end
                1: begin
                        if (LCD_ready) begin//estado de espera de termino de envio a LCD
                                state <= nextstate; //se sigue procedimiento
                        end
                end
                2: begin
                        msg_buffer <= ASCII_msg; //envio de mensaje de texto ACSII a LCD
                        show <= 1;

                        if (mode_B) begin       //si bit mode_B esta activo, se envia ademas numero
                                nextstate <= 3;
                        end
                        else begin
                                nextstate <= 6;  //se termina procedimiento
                        end

                        state <= 1;             //se salta a estado de espera
                end
                3: begin
                        msg_buffer <= 48'h1B5B313B3048;         //envia salto a segunda linea
                        show <= 1;

                        if (signo && mode_C) begin
                                nextstate <= 4;  //si se imprime signo - se va a estado 4
                        end
                        else begin
                                nextstate <= 5;  //de lo contrario se imprime en LCD sin signo
```

```
                        end
                        state <= 1;
                    end
                    4: begin
                        msg_buffer <= "-";          //envia signo -
                        show <= 1;
                        nextstate <= 5;             //se sigue mostrando numero
                        state <= 1;
                    end
                    5: begin
                        if (conversion_ready) begin //si la conversion esta lista, se envia a pantalla
                            msg_buffer <= ACSIInumber;
                            show <= 1;
                            nextstate <= 6;     //se termina procedimiento
                            state <= 1;
                        end
                    end
                    6: begin
                        ready <= 1;
                        state <= 0;
                    end
                    default: begin
                        state <= 0;
                    end
                endcase
            end


endmodule
```

A.2.q Herramientas – conversor binario a ASCII

```
`timescale 1ns / 1ps
module converter2(binarynumber, clk, start, ready, ACSIInumber);

    input [31:0] binarynumber;
    input clk;
    input start;

    output wire [11*8-1:0] ACSIInumber; //10 Caracteres maximo para conversion, uno vacio para fin mensaje
    output reg ready = 1;

    reg [3:0] digits;
    reg [1:0] state = 0;
    reg [4:0] pointer;
    reg [4:0] bitcounter;
    reg [4:0] length;
    reg MODIN;

    wire [3:0] Q0, Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9;
    reg [3:0] D0, D1, D2, D3, D4, D5, D6, D7, D8, D9;
    reg init;

    assign ACSIInumber[7:0] = (digits >= 1) ? {4'd3,D0} : 8'h30;
    assign ACSIInumber[15:8] = (digits >= 2) ? {4'd3,D1} : 8'h00;
```

```
assign ACSIInumber[23:16] = (digits >= 3) ? {4'd3,D2} : 8'h00;
assign ACSIInumber[31:24] = (digits >= 4) ? {4'd3,D3} : 8'h00;
assign ACSIInumber[39:32] = (digits >= 5) ? {4'd3,D4} : 8'h00;
assign ACSIInumber[47:40] = (digits >= 6) ? {4'd3,D5} : 8'h00;
assign ACSIInumber[55:48] = (digits >= 7) ? {4'd3,D6} : 8'h00;
assign ACSIInumber[63:56] = (digits >= 8) ? {4'd3,D7} : 8'h00;
assign ACSIInumber[71:64] = (digits >= 9) ? {4'd3,D8} : 8'h00;
assign ACSIInumber[79:72] = (digits == 10) ? {4'd3,D9} : 8'h00;
assign ACSIInumber[87:80] = 8'h00;

bin2bcd digito0(Q0, MODOUT0, MODIN, init, clk);
bin2bcd digito1(Q1, MODOUT1, MODOUT0, init, clk);
bin2bcd digito2(Q2, MODOUT2, MODOUT1, init, clk);
bin2bcd digito3(Q3, MODOUT3, MODOUT2, init, clk);
bin2bcd digito4(Q4, MODOUT4, MODOUT3, init, clk);
bin2bcd digito5(Q5, MODOUT5, MODOUT4, init, clk);
bin2bcd digito6(Q6, MODOUT6, MODOUT5, init, clk);
bin2bcd digito7(Q7, MODOUT7, MODOUT6, init, clk);
bin2bcd digito8(Q8, MODOUT8, MODOUT7, init, clk);
bin2bcd digito9(Q9, MODOUT9, MODOUT8, init, clk);

always  @(negedge clk) begin
        case (state)
                0: begin
                        if (start) begin

                                digits <= 0;
                                pointer <= 31;
                                length <= 0;
                                bitcounter <= 0;
                                init <= 0;

                                ready <= 0;
                                state <= state + 1;
                        end
                end
                1: begin
                        if (binarynumber[pointer] == 1'b0) begin
                                if (pointer == 0) begin
                                        ready <= 1;
                                        state <= 0;
                                end
                                else pointer <= pointer - 1;
                        end
                        else begin
                                length <= pointer;
                                bitcounter <= pointer;

                                state <= state + 1;
                        end

                end
                2: begin

                        if (binarynumber < 10)      digits <= 1;
                        else if (binarynumber < 100) digits <= 2;
                        else if (binarynumber < 1000) digits <= 3;
```

```
                        else if (binarynumber < 10000) digits <= 4;
                        else if (binarynumber < 100000) digits <= 5;
                        else if (binarynumber < 1000000) digits <= 6;
                        else if (binarynumber < 10000000) digits <= 7;
                        else if (binarynumber < 100000000) digits <= 8;
                        else if (binarynumber < 1000000000) digits <= 9;
                        else digits <= 10;

                        init <= 1;
                        MODIN <= binarynumber[bitcounter];
                        state <= state + 1;
                end
                3: begin
                        if (bitcounter == 0) begin
                                D0 <= Q0;
                                D1 <= Q1;
                                D2 <= Q2;
                                D3 <= Q3;
                                D4 <= Q4;
                                D5 <= Q5;
                                D6 <= Q6;
                                D7 <= Q7;
                                D8 <= Q8;
                                D9 <= Q9;

                                ready <= 1;
                                state <= 0;
                        end
                        else begin
                                MODIN <= binarynumber[bitcounter-1];
                                bitcounter <= bitcounter - 1;
                        end

                end

                default: begin
                        state <= 0;
                end
        endcase
    end

endmodule
```

A.2.r Herramientas - número binario a BCD

```
`timescale 1ns / 1ps
module bin2bcd(Q, MODOUT, MODIN, INIT_BAR, CLK);
   output reg [3:0] Q;
   output MODOUT;
   input MODIN;
   input INIT_BAR;
   input CLK;

        assign MODOUT = (Q < 5) ? 0 : 1;
```

```
	always  @(posedge CLK) begin
		if (!INIT_BAR)
			Q <= 4'd0;
		else begin
			case (Q)
				4'd5: Q <= {3'd0, MODIN};
				4'd6: Q <= {3'd1, MODIN};
				4'd7: Q <= {3'd2, MODIN};
				4'd8: Q <= {3'd3, MODIN};
				4'd9: Q <= {3'd4, MODIN};
				default: Q <= {Q[2:0], MODIN};
			endcase
		end
	end

endmodule
```

A.2.s Herramientas - transmisor mensaje a LCD

```
`timescale 1ns / 1ps
module displaymsg(msg_buffer, clk, show, ready, TxD_LCD);
	parameter msgwidth = 16*8; //16 Caracteres, 1 byte por caracter

	input clk, show;
	input [msgwidth-1:0] msg_buffer; //Se declara buffer de mensaje con largo de 18 caracteres
	output TxD_LCD;
	output reg ready = 1;

	reg [7:0] msg_send;			//Se declara variable que recibe caracter a caracter
	reg [4:0] contador = 0;
	reg lock = 0;
	reg [4:0] i = 0;
	reg [7:0] TxD_data;
	reg start = 0;
	wire TxD_ready;
	reg [7:0] pointer;
	reg msbbit;
	reg assignation = 0;

	initial
	begin
		start = 0;
	end

	Rs232_T_LCD transmite(TxD_LCD, TxD_ready, clk, start, TxD_data);

	always  @(negedge clk)
	begin
		if(~TxD_ready)   start = 0;

		if (show || lock) //inicio de conteo de longitud de mensaje
		begin
			ready = 0;
			lock = 1;
```

```
                        {msbbit,pointer} = contador*8;

                        if ( msg_buffer[pointer+:8] != 8'b00000000 )
                        begin
                                contador = contador + 1;
                        end
                        else //si se llega al final del mensaje
                        begin
                                lock = 0;
                                assignation = 0; //asignado, probar funcionamiento
                        end
                end
                else if (~lock && ~ready)
                begin
                        if (~assignation)
                        begin
                                i = contador;
                                contador = 0;
                                assignation = 1;
                        end

                        if ( i != 0)
                        begin
                                if(TxD_ready)
                                begin
                                        TxD_data = msg_buffer[(i*8-1)-:8];
                                        start = 1;
                                        i = i - 1;
                                end
                        end
                        else begin
                                if(TxD_ready)
                                begin
                                        ready = 1;
                                        assignation = 0;
                                end
                        end
                end
        end

endmodule
```

A.2.t Herramientas - enlace serial a LCD

```
`timescale 1ns / 1ps
module Rs232_T_LCD(TxD, TxD_ready, Clk, TxD_start, TxD_data);
        input Clk, TxD_start;
        input [7:0] TxD_data;
        output TxD_ready;
        output reg TxD = 1;

        reg [12:0] BaudDivCnt = 0;
        reg [3:0] state = 0;
        reg [7:0] TxD_dataD;
        reg Baud;
```

```
assign TxD_ready = (state == 0) && ~TxD_start;

initial begin
	BaudDivCnt = 0;
	state = 0;
end

always  @(negedge Clk) begin
	if (Baud) begin
		BaudDivCnt <= 0;
		Baud <= 0;
	end
	else if (BaudDivCnt == 13'b1010001011000)
		Baud <= 1;
	else
		BaudDivCnt <= BaudDivCnt + 1;
end

always  @(posedge Clk) begin
	case(state)
		4'd0:
		begin
			if (TxD_start) begin
				TxD_dataD = TxD_data;
				state = state + 1;
			end
			else begin
				TxD = 1; // Line Available
				state = 0;
			end
		end
		4'd1:
		begin
			if (Baud) begin
				TxD = 0; // Start bit?
				state = state + 1;
			end
		end
		4'd2,
		4'd3,
		4'd4,
		4'd5,
		4'd6,
		4'd7,
		4'd8,
		4'd9:
		begin
			if (Baud) begin
				TxD = TxD_dataD[0];
				TxD_dataD = TxD_dataD >> 1;
				state = state + 1;
			end
		end
		4'd10:
		begin
			if (Baud) begin
```

```
                                        TxD = 1; // Stop Bit
                                        state = 0;
                                end
                        end
                        default:
                        begin
                                TxD = 1;
                                state = 0;
                        end
                endcase
        end
endmodule
```

A.2.u Herramientas - driver serial PC

```
`timescale 1ns / 1ps
module send_number_serial(number, clk, start, ready, outserial);
        input [17:0] number;
        input clk;
        input start;

        output reg ready = 1;
        output outserial;

        reg [7:0] data_send;
        reg [1:0] state = 0;

        wire ready_serial, outserial;
        reg start_serial = 0;

        Rs232_Trans envio(outserial, ready_serial, clk, start_serial, data_send, 1'b1);

        always  @(posedge clk) begin
                if (~ready_serial) start_serial = 0;

                case (state)
                        0: begin
                                if (start) begin      //si start es seteado, se envian los 8 bits menos significativos
                                        data_send = number[7:0];
                                        start_serial = 1;

                                        ready = 0;

                                        state = state + 1;
                                end
                        end
                        1: begin
                                if (ready_serial) begin
                                        data_send = number[15:8];
                                        start_serial = 1;

                                        state = state + 1;
                                end
                        end
                        2: begin
```

```
                                        if (ready_serial) begin
                                                data_send = { 6'b0 , number[17:16] };
                                                start_serial = 1;

                                                state = state + 1;
                                        end
                                end
                                3: begin
                                        if (ready_serial) begin
                                                ready = 1;

                                                state = 0;
                                        end
                                end
                        endcase
                end

endmodule
```

A.2.v Herramientas - driver serial

```
`timescale 1ns / 1ps
module Rs232_Trans(TxD, TxD_ready, Clk, TxD_start, TxD_data, parity);
input Clk, TxD_start, parity;
input [7:0] TxD_data;
output TxD, TxD_ready;

reg TxD,parityD;
reg [8:0] BaudDivCnt = 0;
reg [3:0] state = 0;
reg [7:0] TxD_dataD;
reg Baud;

assign TxD_ready = (state == 0) && ~TxD_start;

always  @(posedge Clk) begin
        if (Baud) begin
                BaudDivCnt <= 0;
                Baud <= 0;
        end
        else if (BaudDivCnt == 9'b110110010)
                Baud <= 1;
        else
                BaudDivCnt <= BaudDivCnt + 1;
end

always  @(negedge Clk) begin
        case(state)
                4'd0:
                begin
                        if (TxD_start) begin
                                TxD_dataD = TxD_data;
                                parityD = parity;
                                state = state + 1;
                        end
```

```
                        else begin
                                TxD = 1; // Line Available
                                state = 0;
                        end
                end
                4'd1:
                begin
                        if (Baud) begin
                        TxD = 0; // Start bit?
                        state = state + 1;
                        end
                end
                4'd2,
                4'd3,
                4'd4,
                4'd5,
                4'd6,
                4'd7,
                4'd8,
                4'd9:
                begin
                        if (Baud) begin
                        TxD = TxD_dataD[0];
                        TxD_dataD = TxD_dataD >> 1;
                        state = state + 1;
                        end
                end
                4'd10:
                begin
                        if (Baud) begin
                        TxD = parityD;
                        state = state + 1;
                        end
                end
                4'd11:
                begin
                        if (Baud) begin
                        TxD = 1;
                        state = 0;
                        end
                end
                default:
                begin
                        TxD = 1;
                        state = 0;
                end
        endcase
end

endmodule
```

A.3 Interfaz Comunicación Serial

La interfaz aquí mostrada, despliega una simple ventana que maneja la comunicación serial con la FPGA, de esta recibe números de 18 bits y los almacena con signo en un archivo especificado en código fuente (ResultadosFPGA.txt). Una captura de pantalla muestra el simple manejo de la ventana en la Fig. A.1.

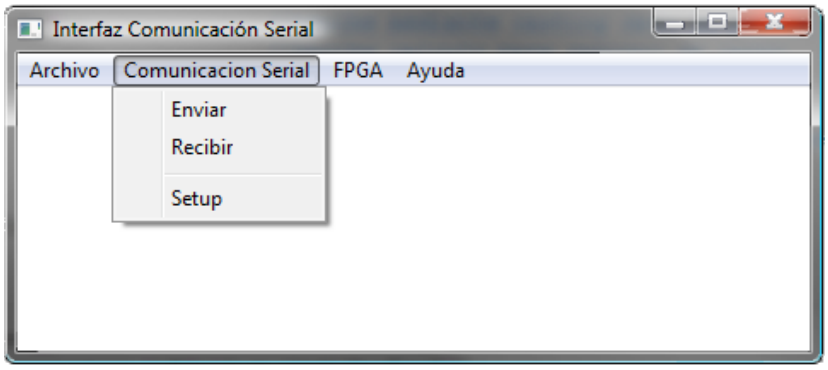


Fig. A.1: Interfaz Comunicación Serial

En los siguientes apartados se presentan los encabezados y archivos de proyecto relacionados con la programación de esta interfaz.

A.3.a Archivo Interfaz_serial.c

```
#include "interfaz_serial.h"

int msgcounter = 0;                 //contador de mensajes entrantes

int characters_per_number = 3;      //cantidad de caracteres para formar numero
int numbers = 5120;                 //cantidad de numeros a recibir
int envio_total;                    //Calcula cantidad de datos a recibir
int bits = 18;                      //bits de dato a conversion
FILE *puntero2;                     //puntero de archivo para escribir resultados de FPGA

unsigned int recepcion[4];          //recibe mediante casting dato de puerta serial
int module_result;                  //recibe calculo para retorno de carro
int x,y,z;
long int entero;
char buffer[10];                    //recibe el numero para convertir a entero con signo y escribir a archivo
int warning = 0;

void HandleASuccessfulRead(DWORD dwRead)
{
```

```
	if (dwRead)
	{
		msgcounter += 1;
		recepcion[(msgcounter-1)%3+1] = (int)MsgRcv[0];

		if (File_open2)
		{

			module_result = msgcounter%characters_per_number;
			 if (module_result == 0)
			{
				x = recepcion[3];
				y = recepcion[2];
				z = recepcion[1];

				entero = x*65536 + y*256 + z;

				if (entero >= pow(2,bits-1))
				{
					entero = -(pow(2,bits)-entero);
				}
				sprintf(buffer,"%d\n",entero);
				fputs(buffer,puntero2);

				if (msgcounter == envio_total)
				{
					MessageBoxPrintf(NULL,MB_OK | MB_ICONINFORMATION,
"Terminado", "Se ha terminado de enviar datos, se cierra archivo");
					fclose(puntero2);
					File_open2 = FALSE;
				}

			}
		}
		else
		{
			MessageBoxPrintf(NULL,MB_OK|MB_ICONINFORMATION,"Error", "No se ha abierto
archivo para recepcion");
		}
	}

	return;
}

BOOL LoadFile(HWND hEdit, LPSTR pszFileName)
{
  HANDLE hFile;
  BOOL bSuccess = FALSE;

  hFile = CreateFile(pszFileName, GENERIC_READ, FILE_SHARE_READ, NULL,
    OPEN_EXISTING, 0, 0);
  if(hFile != INVALID_HANDLE_VALUE)
  {
	DWORD dwFileSize;
    dwFileSize = GetFileSize(hFile, NULL);
    if(dwFileSize != 0xFFFFFFFF)
    {
```

```
      LPSTR pszFileText;
      pszFileText = (LPSTR)GlobalAlloc(GPTR, dwFileSize + 1);
      if(pszFileText != NULL)
      {
        DWORD dwRead;
        if(ReadFile(hFile, pszFileText, dwFileSize, &dwRead, NULL))
        {
          pszFileText[dwFileSize] = 0; // Null terminator
          if(SetWindowText(hEdit, pszFileText))
            bSuccess = TRUE; // It worked!
        }
        GlobalFree(pszFileText);
      }
    }
    CloseHandle(hFile);
  }
  return bSuccess;
}

BOOL SaveFile(HWND hEdit, LPSTR pszFileName)
{
  HANDLE hFile;
  BOOL bSuccess = FALSE;

  hFile = CreateFile(pszFileName, GENERIC_WRITE, 0, 0,
    CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, 0);
  if(hFile != INVALID_HANDLE_VALUE)
  {
    DWORD dwTextLength;
    dwTextLength = GetWindowTextLength(hEdit);
    if(dwTextLength > 0)
    {
      LPSTR pszText;
      pszText = (LPSTR)GlobalAlloc(GPTR, dwTextLength + 1);
      if(pszText != NULL)
      {
        if(GetWindowText(hEdit, pszText, dwTextLength + 1))
        {
          DWORD dwWritten;
          if(WriteFile(hFile, pszText, dwTextLength, &dwWritten, NULL))
            bSuccess = TRUE;
        }
        GlobalFree(pszText);
      }
    }
    CloseHandle(hFile);
  }
  return bSuccess;
}

BOOL DoFileOpenSave(HWND hwnd, BOOL bSave)
{
  OPENFILENAME ofn;
  char szFileName[MAX_PATH];

  FILE *puntero;
  char stemp[11];
```

```
  ZeroMemory(&ofn, sizeof(ofn));
  szFileName[0] = 0;

  ofn.lStructSize = sizeof(ofn);
  ofn.hwndOwner = hwnd;
  ofn.lpstrFilter = "Text Files (*.txt)\0*.txt\0All Files (*.*)\0*.*\0\0";
  ofn.lpstrFile = szFileName;
  ofn.nMaxFile = MAX_PATH;
  ofn.lpstrDefExt = "txt";

  if(bSave)
  {
    ofn.Flags = OFN_EXPLORER | OFN_PATHMUSTEXIST | OFN_HIDEREADONLY |
      OFN_OVERWRITEPROMPT;

    if(GetSaveFileName(&ofn))
    {
      if(!SaveFile(GetDlgItem(hwnd, IDC_MAIN_TEXT), szFileName))
      {
        MessageBox(hwnd, "Save file failed.", "Error",
          MB_OK | MB_ICONEXCLAMATION);
        return FALSE;
      }
    }
  }
  else
  {
    ofn.Flags = OFN_EXPLORER | OFN_FILEMUSTEXIST | OFN_HIDEREADONLY;
    if(GetOpenFileName(&ofn))
    {
       MessageBoxPrintf(NULL,MB_OK|MB_ICONINFORMATION,"Archivo..", "archivo en %s", szFileName);

       puntero = fopen(szFileName,"w");

       if(puntero == NULL)
       {
        MessageBoxPrintf(NULL,MB_OK|MB_ICONINFORMATION,"Aviso"," Existen problemas para cargar
archivo");
       }
       else
       {  // Trabajando con archivo creado anteriormente

        File_open = TRUE;

        fclose(puntero);          //Se cierra el archivo (libera puntero)

       } //fin archivo

      /*if(!LoadFile(GetDlgItem(hwnd, IDC_MAIN_TEXT), szFileName))
      {
        MessageBox(hwnd, "Load of file failed.", "Error",
          MB_OK | MB_ICONEXCLAMATION);
        return FALSE;
      }*/
    }
  }
```

```
  return TRUE;
}

BOOL CALLBACK SendDlgProc(HWND hwnd, UINT Message, WPARAM wParam, LPARAM lParam)
{
	int		len;
	char	buf[MAX_PATH];

	switch(Message)
	{
		case WM_INITDIALOG:
		{
			SetDlgItemText(hwnd, CD_MSG, "");
		}
		break;
		case WM_COMMAND:
		{
			switch(LOWORD(wParam))
			{
				case CD_SEND:
				{
					len = GetWindowTextLength(GetDlgItem(hwnd, CD_MSG));
					if(len > 0)
					{
						GetDlgItemText(hwnd, CD_MSG, buf, len + 1);
						if (WriteABuffer(buf,len))

	MessageBoxPrintf(NULL,MB_OK|MB_ICONINFORMATION,"Enviado", "Mensaje %s enviado con %i
Bytes",buf,len);
						else

	MessageBoxPrintf(NULL,MB_OK|MB_ICONEXCLAMATION,"No Enviado", "Error Mensaje no se pudo
enviar\n%s", DescribeLastError(GetLastError()));
						EndDialog(hwnd,IDOK);
					}
				}
				break;
			}
		}
		break;
		case WM_CLOSE:
			EndDialog(hwnd,IDCANCEL);
		break;
		default:
			return FALSE;
	}
	return TRUE;
}

BOOL CALLBACK ComDlgProc(HWND hWnd, UINT Message, WPARAM wParam, LPARAM lParam)
{
	int len;
	char buf[MAX_PATH];

	switch(Message)
	{
		case WM_INITDIALOG:
```

```
                {
                        SetDlgItemText(hWnd, CD_COMNUM, "COM1");
                }
                break;
                case WM_COMMAND:
                {
                        switch(LOWORD(wParam))
                        {
                                case CD_SETTING:
                                {
                                        len = GetWindowTextLength(GetDlgItem(hWnd, CD_COMNUM));
                                        if(len > 0)
                                        {
                                                GetDlgItemText(hWnd, CD_COMNUM, buf, len + 1);
                                                buf[len + 1] = '\0';
                                                memcpy(pcCommPort,buf,len+1);
                                                EndDialog(hWnd,IDOK);
                                        }
                                }
                                break;
                        }
                }
                break;
                case WM_CLOSE:
                        EndDialog(hWnd,IDCANCEL);
                break;
                default:
                        return FALSE;
        }
        return TRUE;
}

LRESULT CALLBACK WndProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
        static BOOL Dialog;
        static HANDLE   hThread;
        static DWORD    dwThreadId2;
        static DWORD    dwThreadId1;

        switch(msg)
        {
                case WM_CREATE:
                {
        puntero2 = fopen("emuFPGA.txt","w");
        File_open2 = TRUE;
        envio_total = characters_per_number*numbers;

                }
                break;
                case WM_COMMAND:
                {
                        switch(LOWORD(wParam))
                        {
                                case CM_ENVIAR:
                                {
            if(File_open && serial_setup)
               DialogBox(GetModuleHandle(NULL),MAKEINTRESOURCE(DLG_SEND), hWnd, SendDlgProc);
```

```
            else if (!File_open && !serial_setup)
                MessageBoxPrintf(hWnd,MB_OK,"Error","No se ha configurado la comunicacion serial y no se ha
abierto archivo de datos");
            else if (!File_open)
                MessageBoxPrintf(hWnd,MB_OK,"Error","No se ha abierto archivo de datos, para envio");
            else if (!serial_setup)
                MessageBoxPrintf(hWnd,MB_OK,"Error","No se ha configurado la comunicacion serial");
                                    }
                                    break;
                                    case CM_RECV:
                                    {
                                            if (!RECV)
                                            {
                                                    RECV = TRUE;
                                                    MessageBoxPrintf(hWnd,MB_OK,"RECIBE","Ciclo de
Recepción Iniciado");
                                                    hThread = CreateThread(NULL, 0, RecvProc, 0, 0,
&dwThreadId2);
                                            }
                                            else
                                                    RECV = FALSE;
                                    }
                                    break;
                                    case CM_FPGA_RECV:
            {
                if (!File_open2)
                {
                  puntero2 = fopen("emuFPGA.txt","w");
                  msgcounter = 0;
                  File_open2 = TRUE;
                }
                else
                   MessageBoxPrintf(hWnd,MB_OK,"Error","Archivo ya abierto, inicia envio desde FPGA");
            }
            break;
                                    case CM_ABOUT:

            DialogBox(GetModuleHandle(NULL),MAKEINTRESOURCE(IDD_ABOUT), hWnd, AboutDlgProc);
                                    break;
                                    case CM_HELP:

            DialogBox(GetModuleHandle(NULL),MAKEINTRESOURCE(IDD_HELP), hWnd, HelpDlgProc);
                                    break;
                                    case CM_FILE_OPEN:
                DoFileOpenSave(hWnd, FALSE);
             break;
             case CM_SETUP:
                Dialog = DialogBox(GetModuleHandle(NULL),MAKEINTRESOURCE(DLG_COM), hWnd,
ComDlgProc);
                                if (Dialog)
                                    Settings(hWnd);

             break;
                                    case CM_EXIT:
                                    {
                                            DestroyWindow(hWnd);
                                    }
```

```
                                        break;
                                }
                        }
                        break;
                        case WM_CLOSE:
                        {
                                DestroyWindow(hWnd);
                        }
                        break;
                        case WM_DESTROY:
                        {
                                PostQuitMessage(0);
                        }
                        break;
                        default:
                                return DefWindowProc(hWnd, msg, wParam, lParam);
                        break;
        }

        return 0;
}

BOOL CALLBACK AboutDlgProc(HWND hwnd, UINT Message, WPARAM wParam, LPARAM lParam)
{
  switch(Message)
  {
    case WM_INITDIALOG:

    return TRUE;
    case WM_COMMAND:
      switch(LOWORD(wParam))
      {
        case IDOK:
          EndDialog(hwnd, IDOK);
        break;
        case IDCANCEL:
          EndDialog(hwnd, IDCANCEL);
        break;
      }
    break;
    default:
      return FALSE;
  }
  return TRUE;
}

BOOL CALLBACK HelpDlgProc(HWND hwnd, UINT Message, WPARAM wParam, LPARAM lParam)
{
  switch(Message)
  {
    case WM_INITDIALOG:

    return TRUE;
    case WM_COMMAND:
      switch(LOWORD(wParam))
      {
        case IDOK:
```

```
                EndDialog(hwnd, IDOK);
            break;
            case IDCANCEL:
                EndDialog(hwnd, IDCANCEL);
            break;
        }
    break;
    default:
        return FALSE;
  }
  return TRUE;
}

int WINAPI WinMain(HINSTANCE hInstance, HINSTANCE hPrevInstance,LPSTR lpCmdLine, int nCmdShow)
{
        WNDCLASS WC;
        HWND hWnd;
        MSG Msg;

        g_hInst = hInstance;

        WC.style                = 0;
        WC.lpfnWndProc          = WndProc;
        WC.cbClsExtra           = 0;
        WC.cbWndExtra           = 0;
        WC.hInstance            = hInstance;
        WC.hIcon                = LoadIcon(NULL, IDI_APPLICATION);
        WC.hCursor              = LoadCursor(NULL, IDC_ARROW);
        WC.hbrBackground        = (HBRUSH)(COLOR_WINDOWFRAME);
        WC.lpszMenuName         = MAKEINTRESOURCE(MENU_VENTANA);
        WC.lpszClassName        = "Ventana";

        if(!RegisterClass(&WC))
        {
MessageBoxPrintf(NULL,MB_OK|MB_ICONEXCLAMATION,"ERROR","No se pudo registrar la Ventana");
                return 0;
        }

        hWnd = CreateWindow("Ventana","Interfaz Comunicación Serial" , WS_OVERLAPPEDWINDOW,
CW_USEDEFAULT, CW_USEDEFAULT,480,480,NULL,NULL,hInstance,NULL);

        if(hWnd == NULL)
        {
MessageBoxPrintf(NULL,MB_OK|MB_ICONEXCLAMATION,"ERROR!","No se pudo crear la Ventana");
                return 0;
        }

        ShowWindow(hWnd, nCmdShow);
        UpdateWindow(hWnd);

        while(GetMessage(&Msg, NULL, 0, 0) > 0)
        {
                TranslateMessage(&Msg);
                DispatchMessage(&Msg);
        }

        return Msg.wParam;
```

```
}

char *DescribeLastError(int Error)
{
	LPVOID lpMsgBuf;

	FormatMessage(FORMAT_MESSAGE_ALLOCATE_BUFFER | FORMAT_MESSAGE_FROM_SYSTEM ,
NULL, Error, MAKELANGID(LANG_NEUTRAL, SUBLANG_DEFAULT), (LPTSTR) &lpMsgBuf, 0, NULL );

	return lpMsgBuf;
}
```

A.3.b Archivo Interfaz_serial.h

```
#include <windows.h>
#include <stdio.h>

#define MENU_VENTANA		100

#define  DLG_COM		101
#define DLG_SEND		102
#define IDD_ABOUT		103
#define IDD_HELP		104
#define  CD_COMNUM		1000
#define  CD_SETTING		1001
#define CD_MSG			1002
#define CD_SEND			1003
#define IDC_MAIN_TEXT		1004
#define CM_ENVIAR		40000
#define CM_RECV			40001
#define CM_HELP			40002
#define CM_EXIT			40003
#define CM_FILE_OPEN		40004
#define CM_SETUP		40005
#define CM_ABOUT		40006
#define CM_FPGA_RECV		40007
#define IDC_STATIC		-1

float num;
BOOL RECV,SEND;
char pcCommPort[MAX_PATH];
char MsgSnd[MAX_PATH];
unsigned char MsgRcv[MAX_PATH];
HANDLE hCom;
static HINSTANCE g_hInst = NULL;
BOOL File_open = FALSE;
BOOL File_open2 = FALSE;
BOOL serial_setup = FALSE;

char *DescribeLastError(int Error);
BOOL CALLBACK ComDlgProc(HWND hWnd, UINT Message, WPARAM wParam, LPARAM lParam);
void Settings(HWND hWnd);
BOOL CALLBACK AboutDlgProc(HWND hwnd, UINT Message, WPARAM wParam, LPARAM lParam);
BOOL DoFileOpenSave(HWND hwnd, BOOL bSave);
BOOL LoadFile(HWND hEdit, LPSTR pszFileName);
```

```
BOOL SaveFile(HWND hEdit, LPSTR pszFileName);
BOOL CALLBACK HelpDlgProc(HWND hwnd, UINT Message, WPARAM wParam, LPARAM lParam);
void HandleASuccessfulRead(DWORD dwRead);

int MessageBoxPrintf(HWND hwnd,unsigned int Type, char* szCaption, char* szFormat, ...)
{
	char szBuffer[1024];
	va_list pArgList;
	va_start(pArgList, szFormat);
	_vsnprintf(szBuffer, sizeof(szBuffer), szFormat, pArgList);
	va_end (pArgList);
	return MessageBox(hwnd, szBuffer, szCaption, Type);
}

void Recv(void)
{
	DWORD dwRead = 0;
	DWORD dwRes;
	BOOL fWaitingOnRead = FALSE;
	OVERLAPPED osReader;

	memset(&osReader,0,sizeof(osReader));
	osReader.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);

	if (osReader.hEvent == NULL)
	{
		MessageBoxPrintf (NULL,MB_OK|MB_ICONSTOP,"ERROR","Error en CreateEvent\n%s",
DescribeLastError(GetLastError()));
		return;
	}

	if (!fWaitingOnRead)
	{
		if (!ReadFile(hCom, MsgRcv, 1, &dwRead, &osReader))
		{
			if (GetLastError() != ERROR_IO_PENDING)
			{
				MessageBoxPrintf (NULL,MB_OK|MB_ICONSTOP,"ERROR","Error en
Comunicaciones\n%s", DescribeLastError(GetLastError()));
			}
			else
			fWaitingOnRead = TRUE;
		}
		else
		{
			HandleASuccessfulRead(dwRead);
		}
	}

	if (fWaitingOnRead)
	{
		dwRes = WaitForSingleObject(osReader.hEvent, INFINITE);

		switch(dwRes)
		{
			case WAIT_OBJECT_0:
			{
```

```
                                        if (!GetOverlappedResult(hCom, &osReader, &dwRead, FALSE))
                                        {
                                                MessageBoxPrintf (NULL,MB_OK|MB_ICONSTOP,"ERROR","Error en
Comunicaciones\n%s", DescribeLastError(GetLastError()));
                                        }
                                        else
                                                HandleASuccessfulRead(dwRead);
                                        fWaitingOnRead = FALSE;
                                }
                                break;
                                default:
                                {
                                        MessageBoxPrintf (NULL,MB_OK|MB_ICONSTOP,"ERROR","Error en
WaitForSingleObject\n%s", DescribeLastError(GetLastError()));
                                }
                                break;
                        }
                }
                CloseHandle(osReader.hEvent);
                return;
}

BOOL WriteABuffer(char * lpBuf, DWORD dwToWrite)
{
        OVERLAPPED osWrite;
        DWORD dwWritten;
        DWORD dwRes;
        BOOL fRes;

        memset(&osWrite,0,sizeof(osWrite));
        osWrite.hEvent = CreateEvent(NULL, TRUE, FALSE, NULL);
        if (osWrite.hEvent == NULL)
        {
                MessageBoxPrintf (NULL,MB_OK|MB_ICONSTOP,"ERROR","Error en CreateEvent\n%s",
DescribeLastError(GetLastError()));
                return FALSE;
        }

        if (!WriteFile(hCom, lpBuf, dwToWrite, &dwWritten, &osWrite))
        {
                if (GetLastError() != ERROR_IO_PENDING)
                {
                        MessageBoxPrintf (NULL,MB_OK|MB_ICONSTOP,"ERROR","Error en
Comunicaciones\n%s", DescribeLastError(GetLastError()));
                        fRes = FALSE;
                }
                else
                {
                        dwRes = WaitForSingleObject(osWrite.hEvent, INFINITE);
                        switch(dwRes)
                        {
                                case WAIT_OBJECT_0:
                                {
                                        if (!GetOverlappedResult(hCom, &osWrite, &dwWritten, FALSE))
                                                fRes = FALSE;
                                        else
                                                fRes = TRUE;
```

```
                                        }
                                        break;
                                        default:
                                        {
                                                MessageBoxPrintf (NULL,MB_OK|MB_ICONSTOP,"ERROR","Error en
WaitForSingleObject\n%s", DescribeLastError(GetLastError()));
                                                fRes = FALSE;
                                        }
                                        break;
                                }
                        }
                }
                else
                        fRes = TRUE;
                CloseHandle(osWrite.hEvent);
                return fRes;
}

DWORD WINAPI RecvProc(LPVOID lpParam)
{
        while(RECV)
        {
                Recv();
        }

        return 0;
}

void Settings(HWND hWnd)
{
        char PORTSET[MAX_PATH] = "baud=115200 parity=M data=8 stop=1";
        COMMCONFIG lpCC;

        hCom = CreateFile(pcCommPort, GENERIC_READ | GENERIC_WRITE, 0, NULL, OPEN_EXISTING,
FILE_ATTRIBUTE_NORMAL | FILE_FLAG_OVERLAPPED | FILE_FLAG_NO_BUFFERING, NULL);

        if (hCom == INVALID_HANDLE_VALUE)
        {
                MessageBoxPrintf (NULL,MB_OK|MB_ICONSTOP,"ERROR","Error en CreateFile\n%s",
DescribeLastError(GetLastError()));
                return;
        }

        if (!GetCommState(hCom, &lpCC.dcb))
        {
                MessageBoxPrintf (NULL,MB_OK|MB_ICONSTOP,"ERROR","Error en GetCommState\n%s",
DescribeLastError(GetLastError()));
                return;
        }

        BuildCommDCB(PORTSET,&lpCC.dcb);

        if (!SetCommState(hCom, &lpCC.dcb))
        {
                MessageBoxPrintf (NULL,MB_OK|MB_ICONSTOP,"ERROR","Error en SetCommState\n%s",
DescribeLastError(GetLastError()));
                return;
```

```
        }

        if (!CommConfigDialog(pcCommPort,hWnd,&lpCC))
        {
                MessageBoxPrintf (NULL,MB_OK|MB_ICONSTOP,"ERROR","Error en CommConfigDialog\n%s",
DescribeLastError(GetLastError()));
                return;
        }

        if (!SetCommState(hCom, &lpCC.dcb))
        {
                MessageBoxPrintf (NULL,MB_OK|MB_ICONSTOP,"ERROR","Error en SetCommState\n%s",
DescribeLastError(GetLastError()));
                return;
        }

        MessageBoxPrintf (NULL,MB_OK|MB_ICONINFORMATION,"FINALIZADO","Puerto serial %s
satisfactoriamente reconfigurado", pcCommPort);
        serial_setup = TRUE;
        return;

}
```

A.3.c Archivo Interfaz_serial.rc

```
#include "interfaz_serial.h"

MENU_VENTANA MENU DISCARDABLE
BEGIN
   POPUP "&Archivo"
   BEGIN
      MENUITEM "A&brir...",          CM_FILE_OPEN
      MENUITEM SEPARATOR
      MENUITEM "&Salir",             CM_EXIT
   END
   POPUP "&Comunicacion Serial"
   BEGIN
      MENUITEM "&Enviar",            CM_ENVIAR
      MENUITEM "&Recibir",           CM_RECV
      MENUITEM SEPARATOR
      MENUITEM "Se&tup",             CM_SETUP
   END
   POPUP "&FPGA"
   BEGIN
      MENUITEM "&Reiniciar recepcion",   CM_FPGA_RECV
   END

   POPUP "A&yuda"
   BEGIN
      MENUITEM "U&so de programa",       CM_HELP
      MENUITEM "A&cerca de...",          CM_ABOUT
   END
END

DLG_SEND DIALOG DISCARDABLE  0, 0, 186, 66
```

```
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Send Message"
FONT 8, "MS Sans Serif"
BEGIN
   DEFPUSHBUTTON   "Enviar",CD_SEND,68,37,50,14
   CTEXT           "Escriba el Texto a enviar",-1,49,7,90,8
   EDITTEXT        CD_MSG,55,18,76,12,ES_CENTER | ES_AUTOHSCROLL
END

DLG_COM DIALOG DISCARDABLE  0, 0, 200, 100
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Puerto Serial"
FONT 8, "MS Sans Serif"
BEGIN
   PUSHBUTTON      "Propiedades",CD_SETTING,75,60,50,14
   CTEXT           "Ingrese el Nombre del Puerto",-1,50,20,110,20
   EDITTEXT        CD_COMNUM,60,40,77,13,ES_CENTER | ES_AUTOHSCROLL
END

IDD_ABOUT DIALOG DISCARDABLE  0, 0, 239, 66
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Acerca de..."
FONT 8, "MS Sans Serif"
BEGIN
   DEFPUSHBUTTON   "&OK",IDOK,174,18,50,14
   PUSHBUTTON      "&Cancel",IDCANCEL,174,35,50,14
   GROUPBOX        "Acerca de este programa...",IDC_STATIC,7,7,225,52
   CTEXT           "Programa de manejo de puerta serial con FPGA Virtex-II Pro\r\n\r\npor Daniel Herrera P.",
                   IDC_STATIC,16,18,144,33
END

IDD_HELP DIALOG DISCARDABLE  0, 0, 239, 76
STYLE DS_MODALFRAME | WS_POPUP | WS_CAPTION | WS_SYSMENU
CAPTION "Ayuda"
FONT 8, "MS Sans Serif"
BEGIN
   DEFPUSHBUTTON   "&OK",IDOK,174,18,50,14
   GROUPBOX        "Manejo del programa",IDC_STATIC,7,7,225,62
   CTEXT           "Este software abre un enlace serial con un dispositivo de comunicacion serial para envio/recepcion de
datos, con el fin de ser utilizado para resultados en el emulador de circuitos adaptivos",
                   IDC_STATIC,16,18,145,42
END
```