Red Neuronal Implementada en FPGA

Guillermo Tejada Muñoz, Steven Jesús Zarzosa Chávez, Víctor Hugo Benítez Casma

Facultad de Ingeniería Electrónica y Eléctrica de la Universidad Nacional Mayor de San Marcos, Lima, Perú

***Resumen*— El presente artículo describe la implementación de una Red Neuronal Artificial Multicapa en un dispositivo VLSI (Very Large Scale Integration) denominado FPGA (Field Programmable Gate Array), el entrenamiento de la Red Neuronal y la generación de un archivo conteniendo el código VHDL (Hardware Description Language) de la Red se ha realizado con Lenguaje C++.**

Abstract— This article describes the implementation of a Multi-layer Artificial Neural Network within an integrated circuit VLSI (Very Large Scale Integration) called FPGA (Field Programmable Gate Array), the training of the Neural Net and generating a file containing the code VHDL (Hardware Description Language) of the network has been done with language C++.

Palabras Claves— Redes neuronales, Back-propagation, FPGA, VHDL, C++, MATLAB, XSG, ISE.

I. INTRODUCCIÓN

Las Redes Neuronales Artificiales son sistemas de computación inspirados en el funcionamiento del cerebro humano, están compuestos por una gran cantidad de elementos simples de procesamiento (neuronas) conectados entre sí y que operan de forma masivamente paralela. Las redes neuronales consiguen resolver problemas relacionados con el reconocimiento de patrones, predicción, codificación, clasificación, control, optimización, etc. Por lo que su implementación en hardware es una parte esencial para el desarrollo de estas aplicaciones [1]. El interés de los autores del presente trabajo es dar aplicación futura a la Red Neuronal para la clasificación de señales cardiacas.

En la actualidad, la gran mayoría de las aplicaciones de redes neuronales artificiales, se escriben en un programa secuencial de computadora que simula el entrenamiento y ejecución de la red neuronal, este procedimiento no toma en cuenta el paralelismo natural encontrado en la topología de la red neuronal artificial, sin embargo, la implementación de redes neuronales en hardware puede ayudar a solucionar este inconveniente [2].

A. Red Neuronal Artificial

Es un modelo computacional basado en el funcionamiento de redes neuronales biológicas, también puede ser definida como un procesador paralelo masivamente distribuido construido con unidades simples de procesamiento. Entre las características principales de las Redes Neuronales Artificiales (RNAs) se tiene: No linealidad, adaptabilidad e implementación en VLSI. La RNA está constituida por neuronas artificiales interconectadas, pueden tener la topología mostrado en la figura 1.

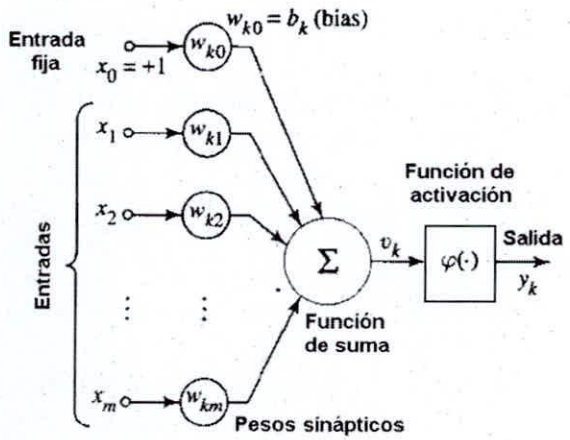

Fig. 1. Modelo de una Red Neuronal Artificial

El modelo fue diseñado por Frank Rosenblatt en 1957, posee entradas con sus respectivos pesos (que determinan la importancia de la entrada), un bloque sumador para las entradas ponderadas y una función de activación que determina la respuesta de la neurona. Una de las entradas tiene valor fijo y peso igual a la

Guillermo Tejada Muñoz, e-mail: gtejadam@unmsm.edu.pe
Steven Jesús Zarzosa Chávez, e-mail:vbenitezc@gmail.com
Víctor Hugo Benítez Casma, e-mail: jesus_zch@yahoo.com

unidad, a ésta se le denomina *Bias*. El tipo de RNA que se ha implementado en el presente trabajo corresponde a una multicapa, completamente conectada, como se observa en la figura 2. Para el entrenamiento (etapa en donde se obtienen los pesos y *Bias* de cada neurona) se utiliza el algoritmo de *Backpropagation*.

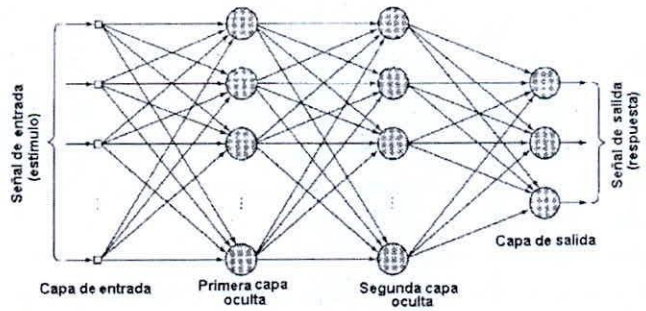

Fig. 2. Red Neuronal Multicapa

B. Algoritmo Backpropagation

Fue diseñado por Paul Werbos en 1974 en su trabajo de tesis en la Universidad de Harvard para obtener el grado de PhD. El entrenamiento descrito por este algoritmo pertenece al aprendizaje de tipo supervisado, pues es necesario un conjunto de datos fijos (salida deseada) para realizar el entrenamiento. Un requerimiento del algoritmo es contar con una función de activación que sea diferenciable, por ese motivo, se ha utilizado la función Sigmoide pues cumple con este requisito y es sencilla para su computación discreta. El algoritmo se describe a continuación [3][4]:

- Se inicializa los valores de los pesos y los Bias para cada neurona de la red, los valores iniciales tienen importancia en medida de que el tiempo de entrenamiento puede ser optimizado.
- Se toma una de las muestras, la cual incluye los valores de las entradas y las salidas deseadas correspondientes, y se colocan sus elementos tanto en la capa de entrada como en la capa de salida.
- Se computa las respuestas de cada neurona por cada capa (EC.1), empezando desde la primera capa hasta llegar a la capa de salida. Además, se obtiene la señal de error (EC.2) para cada neurona en la capa de salida, que es la diferencia entre la salida deseada y la obtenida. A esta etapa se le conoce como propagación hacia adelante.

$$v_{i,j} = bias_{i,j} + \sum_{k=0}^{nnc(i-1)-1} \left(y_{i-1,k} . w_{i,j,k} \right) \quad (1)$$

$$e(n)_j = d(n)_j - y_{nco+1,j} \quad (2)$$

$$y_{i,j} = f\left(v_{i,j}\right) = \frac{1}{1+e^{-v_{i,j}}} \quad (3)$$

- Se computa el gradiente local (EC.4), el cual es la señal de error para las neuronas en la capa de salida, mientras que para las capas ocultas depende de los gradientes y los pesos en la capa siguiente (EC.5). Luego se procede ajustar los pesos (EC.6), los cuales dependen de sus valores anteriores, los gradientes locales, las salidas obtenidas y la tasa de aprendizaje (la cual tiene influencia en el tiempo de entrenamiento). Esta etapa se denomina la propagación hacia atrás.

$$\delta_{nco+1,j} = y_{nco+1,j} . \left(1 - y_{nco+1,j}\right) e(n)_j \quad (4)$$

$$\delta_{i,j} = y_{i,j} . \left(1 - y_{i,j}\right) \sum_{k=0}^{nnc(i+1)-1} \left(\delta_{i+1,k} . w_{i+1,k,j} \right) \quad (5)$$

$$w(n+1)_{i,j,k} = w(n)_{i,j,k} + r_i . y_{i-1,k} . \delta_{i,j} \quad (6)$$

- Se presenta otra de las muestras y luego se realizan las propagaciones hacia adelante y atrás, cuando todas las muestras han sido presentadas se computa el error cuadrático medio (EC.7), el cual depende de las señales de error en cada muestra. Si el error cuadrático medio es menor que un límite determinado, se termina el proceso de aprendizaje, caso contrario, se registra que se ha realizado un *Epoch* y se vuelven a presentar todas las muestras. El valor del límite para el error cuadrático medio determina la exactitud de la respuesta final de la red neuronal.

$$E(n) = \frac{1}{2} \sum_{j=0}^{\#salidas-1} e(n)_j^2 \quad (7)$$

$$MSE = \frac{1}{N} \sum_{n=0}^{N-1} E(n) \quad (8)$$

II. METODOLOGÍA

La figura 3, muestra la metodología que se ha empleado para la implementación de la Red Neuronal Artificial. Una vez definido por el usuario el tamaño de la Red, es decir, el número de entradas, de capas ocultas y de salidas, se procede a entrenarla con un algoritmo de tipo Backpropagation, codificado en Lenguaje C. Concluido el entrenamiento, los pesos y Bias resultantes son utilizados como datos de entrada por otra porción del código C que genera un archivo (*.vhdl) conteniendo el código VHDL (Hardware Description Language) de la estructura de la Red. Finalmente, con el archivo VHDL y empleando la herramienta de software ISE (Integrated Software Environment) se simula y se transfiere el código VHDL a la tarjeta Virtex II Pro, que contiene un FPGA. Para propósitos de demostración, se ha implementado una Red Neuronal que ejecuta la función lógica XOR.

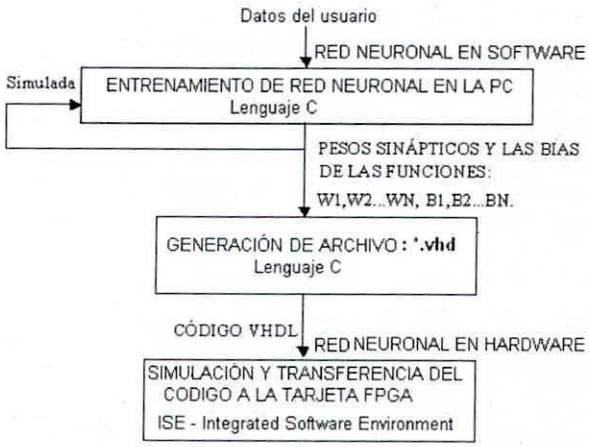

Fig. 3. Metodología para implementar una Red Neuronal prototipo

A. Entrenamiento de la red neuronal

La figura 4, describe el Diagrama de Flujo del programa de entrenamiento *Backpropagation* de la Red Neuronal Multicapa que ha sido codificado en lenguaje C++.

1) Ingreso de datos

Se introducen datos que definan el tamaño de la Red Neuronal como son el número de entradas, salidas, capas y neuronas. Por cada capa, estas últimas se registran en un vector de tamaño variable dependiendo del número de neuronas ingresadas.

Luego se ingresa el número de muestras para el entrenamiento, a continuación se piden ingresar los valores de las entradas y salidas de cada muestra, que se guarda como una fila de una matriz de dos dimensiones.

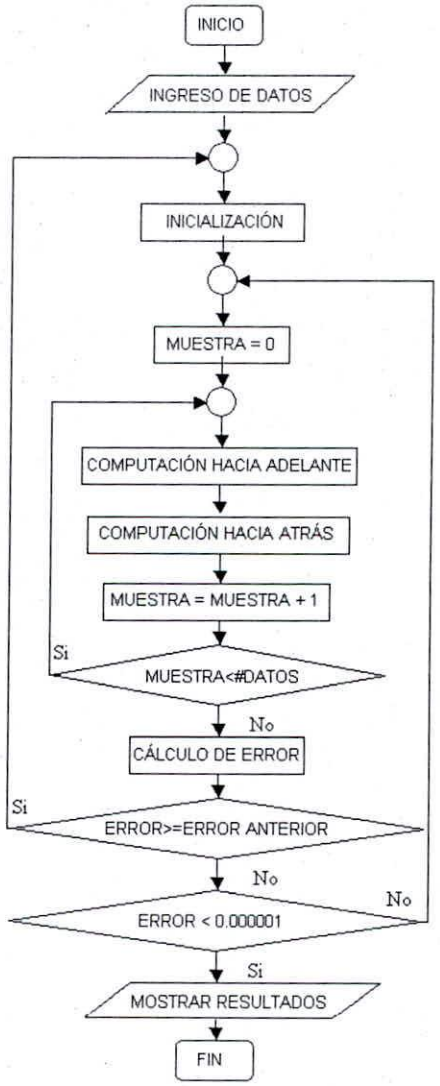

Fig. 4. Diagrama de Flujo del entrenamiento

2) Inicialización

Los pesos y los *Bias* para cada neurona obtienen su valor inicial en forma aleatoria con un límite establecido, este límite o parámetro de entrenamiento empieza con un valor pequeño y será de utilidad para hacer dinámico el aprendizaje de la red. Los *Bias* son almacenados en una matriz de dos dimensiones, donde el número de fila indica la capa en la que se encuentra la neurona y el número de columna indica el número de neurona en esa capa. Los pesos son almacenados en una matriz de tres dimensiones, los dos primeros índices expresan a que neurona pertenece cada peso, análogamente al caso de los *Bias*, con la diferencia que el tercer índice hace referencia al número de la neurona de la capa anterior a la cual está conectada. Tanto para la asignación de pesos y *Bias*, los valores iniciales son configurados con signos intercambiados, esto con la

finalidad de que la media sea cercana a cero, y acelerar la convergencia. La tasa de aprendizaje va disminuyendo de valor a medida que se avanza desde la primera capa hasta la última, para hacer que los valores finales de entrenamiento sean cercanos.

3) Computación hacia adelante

Los valores de las entradas de la muestra actual se ingresan como salidas para la primera capa entonces utilizando los pesos y *Bias* de las neuronas se consigue la respuesta de ellas (que se almacena en una matriz de dos dimensiones con formato de índices igual a la matriz de *Bias*), las cuales son llevadas como entradas para la siguiente capa y se forma un proceso iterativo hasta llegar a la última capa, el procedimiento se representa en la figura 5.

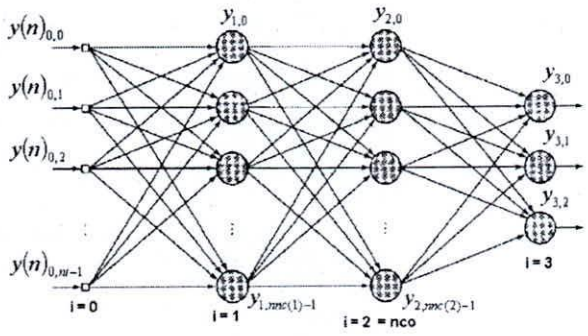

Fig. 5. Propagación hacia delante de entrada a salida

4) Computación hacia atrás

Se obtienen las señales de error en cada salida (diferencias entre la señal deseada y la obtenida) para calcular el error de la muestra, este valor es almacenado como elemento de un vector, para obtener el error cuadrático medio cuando se revisen todas las muestras. Se obtienen los gradientes de las neuronas en la capa de salida y luego se forma un proceso iterativo para obtener los gradientes en las neuronas de las capas anteriores. Seguidamente se modifican los pesos con los valores de los gradientes (almacenados en una matriz de dos dimensiones, con el formato de índices igual que los *Bias*), la respuesta de la neurona en la capa anterior, la tasa de aprendizaje de la capa y el valor actual del peso a ser modificado. El procedimiento se representa en la figura 6.

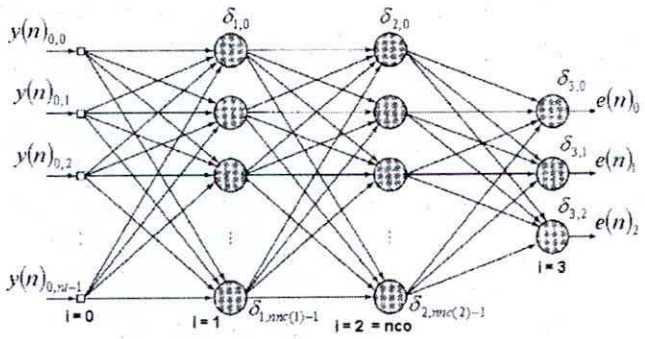

Fig. 6. Propagación hacia atrás de los gradientes localizados.

5) Cálculo de error

Con el vector de error completo se procede a calcular el error cuadrático medio, fundamental para decidir la convergencia de los pesos en el entrenamiento.

6) Mostrar resultados

Se realiza una impresión de los valores de los pesos y los *bias* resultantes del entrenamiento (previamente redondeados con la intención de utilizar menos recursos de hardware a la hora de la implementación), así como el valor final del error cuadrático medio al que se llegó al terminar el entrenamiento.

B. Generación del Código VHDL

Para propósitos de prueba se ha entrenado a la red a fin que responda como las funciones lógicas AND, OR y XOR. La Red Neuronal Multicapa es de dos entradas, dos neuronas en la capa oculta y una neurona en la salida, el tipo de entrenamiento es *Backpropagation.*

Los resultados del programa que son utilizados para la implementación de la red en hardware son los Pesos y los *Bias* de cada neurona. Cómo se puede observar de la figura 7, los resultados de los pesos entre las entradas y las neuronas de la capa oculta son: W100 = 10, W101 = -11, W110 = 11, W111 = -11, entre las neuronas de la capa oculta y de salida son: W200 = -14, W201 = 13. Mientras que los *Bias* para las tres neuronas son respectivamente: B10 = 6, B11 = -5 y B20 = 7. Los Pesos y *Bias* obtenidos son verificados con un segundo programa, inverso al primero, en donde al ingresar estos datos se obtiene la tabla XOR, tal cómo se muestra en la figura 8.

Para generar el código VHDL de la Red Neuronal se tiene dos opciones, la primera consiste en utilizar una herramienta de software denominada Xilinx System

Generator (XSG), que se incorpora a las demás herramientas de Simulink y permite generar el código en VHDL del circuito que se simula de manera automática, y la segunda opción es realizando un programa codificado en lenguaje C++ que puede generar archivos que contienen el código VHDL de la Red Neuronal. Se describirá brevemente ambos métodos.

```
Entrenamiento de una red neuronal multicapa
-------------------------------------------

Numero de entradas: 2
Numero de salidas: 1
Numero de capas ocultas: 1
Numero de neuronas en la capa oculta 1 : 2
Numero de datos disponibles: 4

X0      X1      Y0
0       0       0
0       1       1
1       0       1
1       1       0

Calculando pesos y bias
...
W100 = 10
W101 = -11
B10 = 6

W110 = 11
W111 = -11
B11 = -5

W200 = -14
W201 = 13
B20 = 7

Con los pesos redondeados se obtiene un error de 1.22115e-006

Presione una tecla para continuar . . . _
```

Fig. 7. Resultados de entrenar la Red Neuronal con la función XOR

```
Red neuronal multicapa
----------------------

Numero de entradas: 2
Numero de salidas: 1
Numero de capas ocultas: 1
Numero de neuronas en la capa oculta 1 : 2

Valores de los pesos y bias

W100 = 10
W101 = -11
B10 = 6

W110 = 11
W111 = -11
B11 = -5

W200 = -14
W201 = 13
B20 = 7

La respuesta del sistema es
X0      X1      Y0
0       0       0.00102876
0       1       0.999
1       0       0.997447
1       1       0.0010913

Presione una tecla para continuar . . . _
```

Fig. 8. Validación de los resultados

Parte del procedimiento del primer método, se muestra en la figura 9, en donde se ha implementado la Red Neuronal en Simulink, se ha agregado al circuito mascaras por donde se ingresan los datos del Peso y *Bias* de cada neurona que se calcularon previamente, también en esta figura se observa el bloque XSG incorporado al circuito. Luego de simular el funcionamiento de la red, al activar el bloque XSG, ver figura 10, se abre una ventana en donde se ingresan algunos datos como son: el código de FPGA que se utilizará y la velocidad del *Clock* dentro de la tarjeta FPGA, una vez ingresado estos datos, al presionar el botón "GENERATE" se genera el código VHDL del circuito.

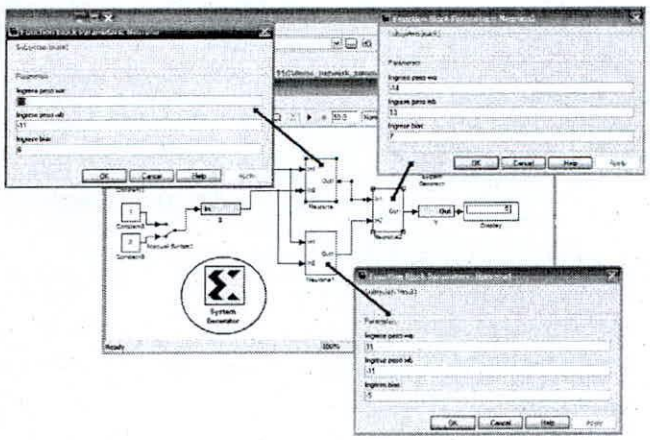

Fig. 9. Red Neuronal en SIMULINK

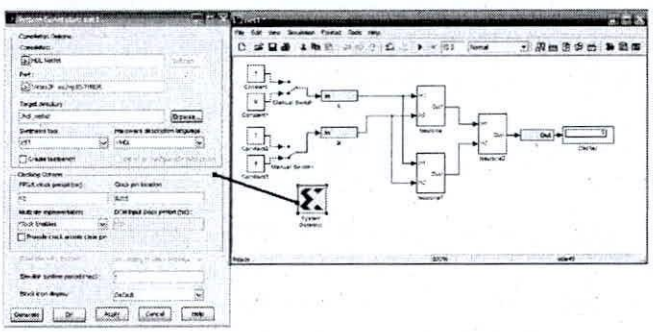

Fig. 10. Generación del código VHDL de la Red Neuronal

La segunda opción, se realiza mediante los programas que se describen en la figura 11. En la secuencia del programa de la figura 11 (a), con los pesos y *bias* calculados en la etapa de entrenamiento se genera un archivo de texto con la extensión .vhd que contiene el código VHDL de las operaciones básicas de una neurona, es decir las de productos entre las entradas y los pesos, de sumas de los productos y de activación de la función de salida, los archivos generados, tienen por nombres desde neurona1.vhd hasta neuronai.vhd, donde i es la última neurona.

En la secuencia del programa de la figura 11(b), con los datos que ingresa el usuario como: nombre del archivo, número de capas de la Red y número de

neuronas por capa, se genera un archivo de texto con el código VHDL que contiene el tamaño y estructura de la Red Neuronal Multicapa deseada, en donde las neuronas se referencian con los nombres genéricos desde neurona1 hasta neuronai. El archivo de texto generado por el programa, conteniendo el código VHDL, tiene el formato: nombre.vhd.

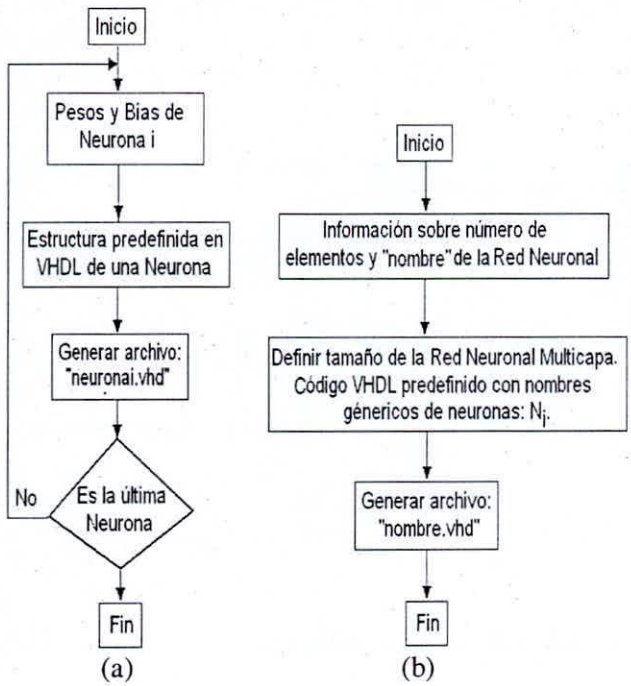

Fig. 11. Diagrama de los programas de generación de código VHDL (a) Para cada neurona (b) Red Neuronal Multicapa.

C. Simulación y Transferencia del código VHD a la tarjeta FPGA

El próximo paso, es pasar el código VHDL, o mejor dicho los archivos generados según se ha descrito anteriormente, a la herramienta de software ISE (Integrated Software Environment), que es desde donde se genera el código binario de los archivos VHDL, el cual se descarga a la tarjeta FPGA, la figura 12 muestra el proceso. La herramienta ISE también permite simular la respuesta de la Red en función al tiempo.

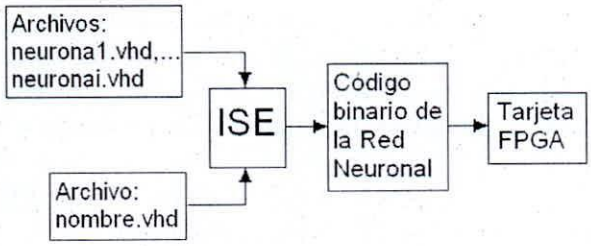

Fig. 12. Proceso de cargar a la tarjeta FPGA

Los pasos que se siguen una vez abierto el programa ISE, son brevemente los siguientes:

- Utilizar el *wizard* para crear un nuevo proyecto, el cual se llamará: OR_Exclusivo_XOR, ver figura 13.
- Adherir los archivos, neurona1.vhd, neurona2.vhd, neurona3.vhd, xor_21_11.vhd, ver figura 14.
- Sintetizar el código, después del cual podemos realizar una vista del esquemático RTL de la Red neuronal.
- Simular el comportamiento de la Red Neuronal, seleccionando la herramienta de simulación ModelSim XE.
- Seleccionamos las variables y periodos de las señales que queremos observar.
- Obtenemos los resultados de la Simulación, ver figura 15.
- En la implementación, se debe realizar la asignación de *pines* para asegurarse de que cada una de las señales externas del FPGA queda conectada al dispositivo electrónico que hemos elegido. En este caso, las conexiones van dirigidas al reloj externo, las entradas y la salida.
- Luego se genera el archivo de programación .bit.
- Finalmente, este último archivo se descarga a la tarjeta ejecutando la opción de Configure Target Device (Impact).

Si se desea consultar con mayores detalles cómo manejar la herramienta ISE para interactuar con una tarjeta FPGA, véase la referencia [5].

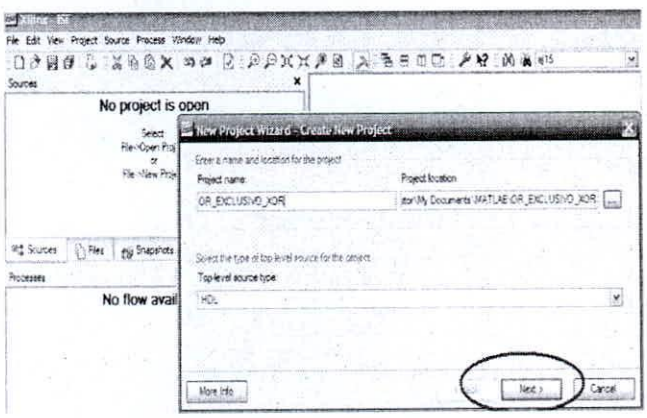

Fig. 13. Creación del proyecto

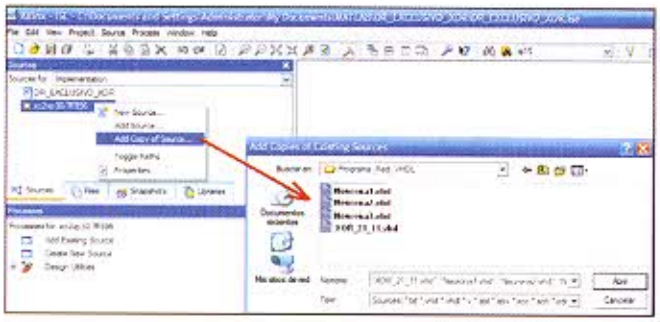

Fig. 14. Adherimos archivos .vhd.

Fig. 15. Resultados de la simulación.

III. CONCLUSIONES

Se ha logrado utilizar un mínimo de herramientas computacionales propietarias al codificar en Lenguaje C++ los programas de entrenamiento y generación de código VHDL de la Red Neuronal Multicapa.

Los programas desarrollados pueden entrenar y generar código VHDL de una Red Neuronal Multicapa de tamaño flexible, el usuario de acuerdo a sus requerimientos define el número de capas y neuronas.

Se ha comprobado que el tiempo de entrenamiento de la red depende de la inicialización de pesos y *bias*, de la tasa de aprendizaje elegida para cada capa, del valor límite del error medio cuadrático y de otros como el factor de correlación, que multiplica el valor de los pesos para una muestra anterior y agrega el resultado en el ajuste de los pesos para la muestra actual.

IV REFERENCIAS

[1] J.C. Moctezuma Eugenio, A. Sánchez Galvez, A. Ata Pérez; *Implementación hardware de Funciones de transferencia para redes neuronales artificiales*, http://www.iberchip.org/iberchip2006/ponencias/103.pdf (consultado junio del 2008).

[2] Ortiz Rodriguez J.M., Martinez Blanco M.R.; *Diseño de neuro-hardware*; http://www.reduaz.mx/eninvie/CD2k6/PDS/04.pdf (consultado junio del 2008)

[3] Haykin, *Simon. Neural Networks A comprehensive foundation*. Prentice Hall, 1999.

[4] Martín, Bonifacio y SANZ, Alfredo. *Redes Neuronales y Sistemas Borrosos*. Alfaomega Grupo Editor S.A., 2007.

[5] Fundatel, Fi-Uner; Práctica 1. *Introducción al diseño con Xilinx ISE*; http://www.uv.es/rosado/dcse/prac1_ISE_VHDL1.pdf (consultado junio del 2008)

[6] Dale, Neil y WEEMS, *Chip. Programación y resolución de Problemas con C++*. McGraw-Hill/Interamericana Editores, S.A., 2007.