NeuroFPGA - Implementando Redes Neuronales Artificiales en Dispositivos Lógicos Programables*

Daniel Ferrer, Ramiro González, Roberto Fleitas, Julio Pérez Acle, Rafael Canetti
{doger,ramg,robfleitas}@adinet.com.uy, {julio,canetti}@fing.edu.uy
Instituto de Ingeniería Eléctrica
Facultad de Ingeniería - UDELAR
Montevideo - Uruguay

8 de Octubre de 2004

Resumen

Se presenta una implementación FPGA de una red neuronal del tipo perceptrón multicapa. El sistema está parametrizado tanto en aspectos relacionados con la red neuronal (e.g.: cantidad de capas y cantidad de neuronas en cada capa) como en aspectos de implementación (e.g.: ancho de palabra, factores de pre-escalado y cantidad de multiplicadores disponibles). Esto permite utilizar el diseño para la realización de diferentes redes, o ensayar diferentes compromisos área-velocidad simplemente recompilando el diseño. Se utilizó aritmética de punto fijo con pre-escalado configurable para cada capa. El sistema fue testeado sobre una placa ARC-PCI de Altera™. Se implementaron varios ejemplos de diferentes dominios de aplicación, mostrando la flexibilidad y facilidad de uso del circuito obtenido. Se obtuvo una aceleración apreciable del algoritmo en comparación con una solución "solo software" basada en el toolbox de Matlab para redes neuronales, incluso a pesar de que la placa utilizada es bastante antigua.

1. Introducción

La implementación en hardware ha estado en uso desde tiempo atrás como un método para acelerar los cálculos de redes neuronales. Las tecnologías usadas van desde custom VLSI (Hammerstrom [7]) hasta lógica programable (e.g.: Arroyo et al. [3], Cox et al. [5], Girau et al. [8]).

Varios de los trabajos anteriores se enfocan a la resolución de un problema en particular, manteniendo duros los parámetros relacionados con la red neuronal (e.g.: Aguilera et al. [4]). Con este enfoque, cada nuevo problema requiere un nuevo esfuerzo de diseño para tomar en cuenta diferencias en los parámetros de la red. Una de las mayores dificultades para obtener una solución general adecuada para un rango amplio de problemas reside en el diseño del autómata que controla el circuito.

En el trabajo presentado aquí se utiliza la reprogramabilidad de los FPGAs para implementar una familia de redes neuronales del tipo perceptrón multicapa, modificando solo un conjunto de parámetros en tiempo de compilación.

El sistema está parametrizado tanto en aspectos relacionados con la red neuronal como en aspectos de implementación. El primer grupo de parámetros (cantidad de capas, neuronas por capa, etc.) se seleccionan en base al problema particular a ser resuelto. El segundo tipo de parámetros se eligen a efectos de optimizar el throughput y la utilización de silicio de acuerdo a las limitaciones impuestas por el dispositivo utilizado. Esto permite el uso del diseño con diferentes tamaños de red y diferentes compromisos área-velocidad simplemente recompilando el diseño.

La función de activación fue construída como una función lineal a tramos, donde las pendientes de cada tramo son potencias negativas de 2. Esta función es también paramétrica y provee una buena aproximación a la función $\tanh(x)$. Otro parámetro permite deshabilitar la función de activación en la capa de salida, permitiendo que los valores de salida de la red no queden acotados por la función de activación.

El diseño obtenido está descrito en AHDL lo que asegura su portabilidad a otros dispositivos lógicos programables de Altera. El circuito fue sintetizado utilizando las herramientas de diseño comerciales de Altera y fue probado sobre una placa ARC-PCI con varios ejemplos de aplicación.

2. Modelo matemático de las redes neuronales

Un perceptrón multicapa consiste en un conjunto de unidades llamadas neuronas interconectadas formando una red. Cada neurona tiene un conjunto de entradas x_i, y una salida y_j; cada entrada es afectada por un coeficiente o peso w_{ji}. El subíndice ij se refiere a la entrada i en la neurona j.

*El presente trabajo es una traducción al español del trabajo [12] presentado por los autores en la conferencia DATE'04 en febrero de 2004.

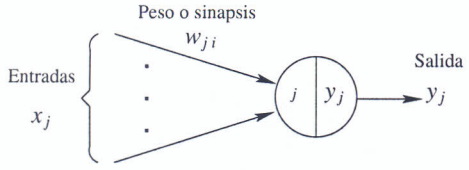

Figura 1: Modelo matemático de una neurona artificial

$$y_j = \varphi(v) = \varphi\Big(\sum_i w_{ji}x_i + b_j\Big) \qquad (1)$$

El cálculo hecho por cada neurona es una suma de productos afectada por una función no lineal, dada por la ecuación 1. La función φ es una función no lineal limitada referida usualmente como función de activación.

Las neuronas se ordenan en capas. Las salidas de las neuronas de cada capa son las entradas de las neuronas de la siguiente capa. Las salidas de la red son producidas por las neuronas en la última capa.

3. Diseño del circuito

La topología de red implementada es una red neuronal "feed-forward" totalmente conectada.

A efectos de obtener una implementación de redes neuronales flexible y escalable se definieron varios parámetros, organizados en dos grupos principales. El primer grupo está relacionado con la arquitectura de la red: la cantidad de capas (`ncapas`), la cantidad de entradas (`nent`), la cantidad de neuronas en cada capa (`nneu1`, `nneu2`, etc.), y la opción de saltear la función de activación en la capa de salida para obtener una salida lineal (`salida_lineal`). Un ejemplo de un conjunto de parámetros de este grupo se muestra en la figura 2 para una red de tres capas.

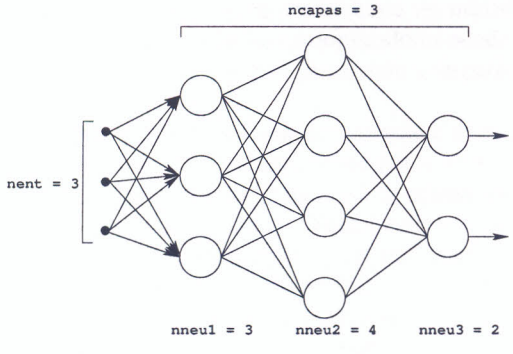

Figura 2: Ejemplo de parámetros para una red de 3 capas

El segundo grupo de parámetros está relacionado con restricciones de implementación: ancho de palabra (`n`), ancho de palabra interno (`l`), cantidad de multiplicadores disponibles (`mx_max /mult`), máximo factor de pre-escalado (`esc_max`), profundidad de pipeline (`pipe_deep`), divisor de frecuencia de reloj (`nclks`) y pendiente de la función de activación (`p`).

Los cálculos se realizan en aritmética de punto fijo y las entradas y pesos son números normalizados al rango $[-1, 1)$ representados en complemento a dos de punto fijo utilizando $n + 1$ bits.

Para un problema dado los patrones de entrada y salida siempre pueden llevarse al intervalo $[-1, 1)$ mediante un cambio de escala. Los pesos y biases de una red entrenada, en cambio, pueden tomar valores fuera de ese intervalo. En ese caso se realiza un pre-escalado fuera de línea de los pesos y biases, compensado por un post-escalado provisto por el circuito a la entrada de la función de activación. El factor de escalado es una potencia entera de dos y debe ser constante para cada capa.

El circuito que realiza la función de activación se presenta más adelante.

3.1. Arquitectura y camino de datos

A efectos de obtener una solución general, incluso bajo restricciones de área, los multiplicadores que caben en el chip se reutilizan tantas veces como sea necesario para obtener el resultado de una neurona completa. El proceso se itera para todas las neuronas de la capa y luego para todas las capas de la red, obteniéndose al final las salidas de la red completa.

Los pesos y biases son utilizados en el cálculo para cada patrón de entradas. Se almacenan en la memoria RAM interna del chip. A cada multiplicador se le asocia un bloque de RAM (RAM de pesos) para almacenar los pesos correspondientes a las sinapsis calculadas por ese multiplicador. Estos pesos se almacenan al comienzo de la operación del circuito y luego el bloque de RAM es operado como una cola circular.

En forma análoga otro bloque de RAM interna (RAM de biases) se utiliza para almacenar los biases de todas las neuronas del circuito.

La placa utilizada tiene dos chips de memoria RAM conectados a buses independientes (bus 1 y bus 2) que se utilizan para almacenar las entradas y salidas de la red respectivamente. Para iniciar los cálculos correspondientes a un conjunto de patrones de entrada, esos patrones se cargan por el computador host en el chip de RAM conectado al bus 1.

Otros dos bloques de registros se implementan internamente en el chip FPGA (almacenamiento temporal REG 1 y REG 2 en figura 3) y son utilizados en forma alternada para almacenar las entradas y salidas de la neurona respectivamente. En el cálculo de una capa uno de los bloques contiene las entradas a la capa. A medida que se van obteniendo las salidas, estas se van almacenando en el otro bloque. Para el cálculo de la siguiente capa los resultados recién obtenidos se utilizan como entradas, y el bloque que originalmente guardaba las entradas se utiliza para almacenar las salidas de la nueva capa. Cuando se obtienen los resultados de la última capa, estos se escriben directamente en la memoria externa a través del bus 2, y en forma simultánea el próximo juego de entradas se lee desde memoria externa a través del bus 1.

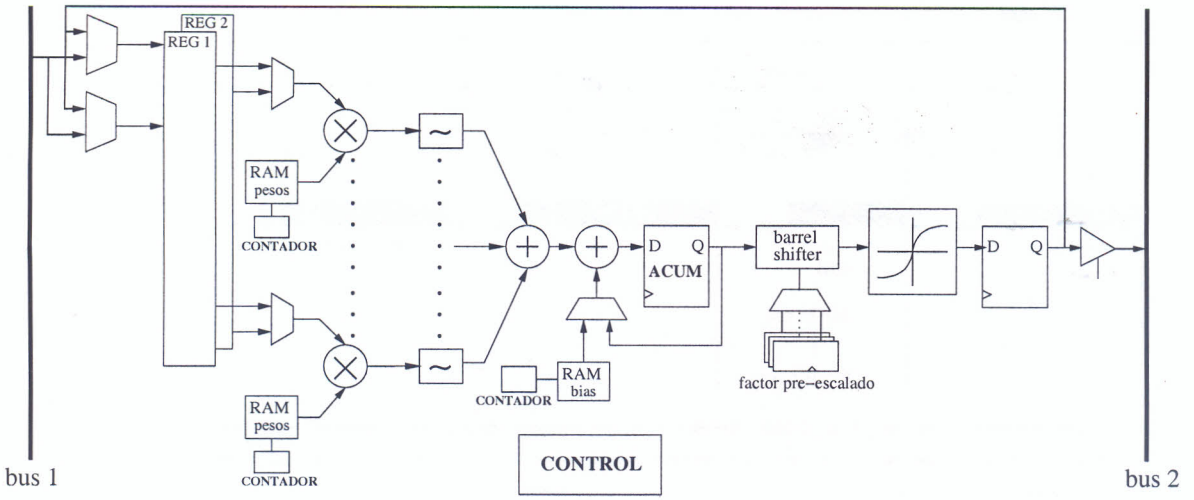

Figura 3: Diagrama de bloques del circuito

Los multiplicadores reciben como entradas los datos desde la memoria RAM de pesos y una de las memorias RAM de almacenamiento temporal. Para evaluar cada neurona, en la pasada inicial se suma a la salida del multiplicador el contenido de la memoria RAM de biases y el resultado se almacena en el registro acumulador. En caso que la neurona tenga más entradas que la cantidad de multiplicadores disponibles, se necesitan pasadas adicionales sumando el contenido del acumulador (de ahí su nombre) en lugar del bias.

Cuando todas las entradas fueron sumadas la salida de la neurona puede obtenerse a través del bloque de la función de activación conectado a la salida del acumulador.

En la figura 5 (sección 3.3) se muestra la temporización de este proceso para un ejemplo de una red de tres capas.

Las entradas y pesos se representan en aritmética de punto fijo de $n + 1$ bits. Cuando se realizan las multiplicaciones el resultado puede crecer hasta un máximo de $2n + 1$ bits. La salida de los multiplicadores puede truncarse a $l+1$ bits, con $n \leq l \leq 2n$, siendo l un parámetro a elegir.

Antes de evaluar la función de activación, el escalado opcional realizado previamente sobre pesos y biases debe ser compensado multiplicando por la misma potencia entera de dos utilizada fuera de línea (realizando un shift a la izquierda del número) para todas las neuronas de la capa. Esto es realizado utilizando un barrel shifter.

La lógica combinatoria que evalúa la suma aritmética de las multiplicaciones es el camino crítico en los retardos del circuito. Por ese motivo se la equipó con un pipeline de profundidad configurable de manera de reducir el retardo máximo entre dos registros y permitir así operar a frecuencias mayores.

3.2. Unidad de control

La red neuronal es controlada principalmente por una máquina de estados cuyo diagrama simplificado se muestra en la figura 4.

Una vez configurado el dispositivo FPGA, el cirtuito "nace" en el estado *inactivo*, a la espera de una señal de comando para iniciar la configuración de la red (estado *inicializacion*) o para iniciar la evaluación de un conjunto de patrones de entrada (estado *calculo*). Estos comandos son suministrados por el computador host escribiendo en direcciones reservadas de memoria (señales de entrada: `start_inic` o `start_calc`).

Al entrar al estado *inicializacion*, el circuito pide acceso al bus 1, y una vez obtenido, lee en secuencia a partir de la dirección 0 (una palabra cada vez, una palabra en cada dirección de memoria) los pesos, biases y factores de escalado para cada capa. El circuito almacena estos valores leídos respectivamente en la RAM de pesos, la RAM de biases y registros dedicados. Una vez completada esta etapa el circuito retorna al estado *inactivo*.

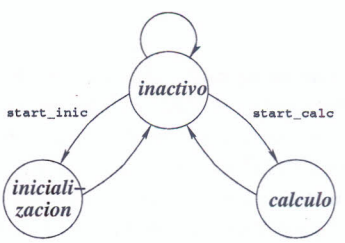

Figura 4: Diagrama de estados simplificado del circuito

En el estado *calculo* en cambio, el circuito solicita ambos buses y cuando los obtiene comienza a leer desde la memoria conectada al bus 1 la cantidad de patrones de entrada a procesar (en la dirección 0) y el primero de esos patrones. Despues que este primer patrón es procesado el circuito comienza el proceso de iteración explicado en la sección anterior. Esta iteración se repite hasta que se procesan la cantidad de juegos de entrada almacenados en la memoria, retornando luego al estado inactivo.

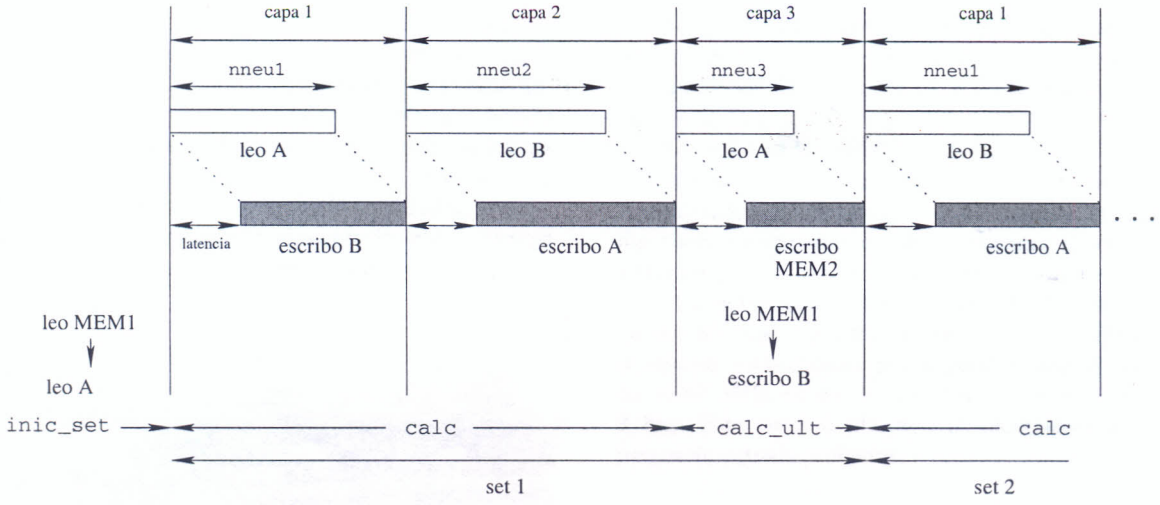

Figura 5: Diagrama de tiempos simplificado del circuito

3.3. Diagrama de Tiempos

La figura 5 muestra un ejemplo de la operación del circuito para una red de 3 capas, con `nneu1`, `nneu2` y `nneu3` neuronas en cada capa respectivamente. El estado *calculo* en el diagrama simplificado de la figura 4 se divide en dos subestados: `calc` y `calc_ult`. Las capas 1 y 2 son procesadas en el estado `calc` y la última capa en `calc_ult`. Las dos memorias RAM de almacenamiento temporal descritas anteriormente se muestran como "A" y "B".

3.4. Función de Activación

Se obtuvo una función de activación con aspecto sigmoidal, que aproxima muy bien a la siguiente función:

$$\varphi(v,a) = \frac{1 - e^{-av}}{1 + e^{-av}} \qquad (2)$$

Como en Aguilera et al. [4], la función implementada es lineal a tramos. Sin embargo, en este trabajo, las pendientes de cada segmento son potencias negativas consecutivas de 2. Esta función se sintetiza como un circuito combinatorio sencillo optimizando el retardo entrada-salida. Todas las operaciones se realizan utilizando "barrel shifters" y compuertas lógicas AND-OR.

La aproximación obtenida se puede observar en la figura 6.

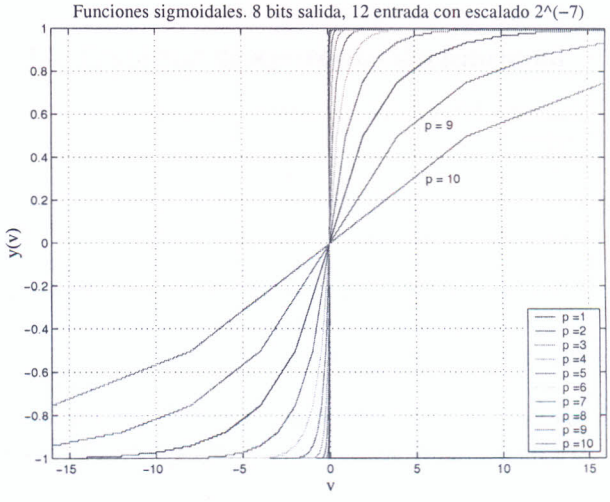

Figura 6: Familia de funciones sigmoidales obtenidas

4. Hardware utilizado

Se sintetizaron y testearon diversos tipos de redes así como algunos ejemplos de aplicación para validar el diseño. El hardware utilizado fue una placa ARC-PCI de Altera. Esta placa tiene una interfaz PCI con un PC y posee 3 chips FPGA EPF10K50. Cada chip contiene 2880 celdas lógicas (50,000 compuertas) y 20 Kbits de memoria embebida en el chip (EABs).

Esta tarjeta tiene además 4 bancos de memoria estática que puede ser accedidos desde los chips FPGA. Uno de los 3 chips está directamente conectado al bus PCI de la tarjeta y cumple la función de interfaz PCI y de árbitro de acceso a los buses de memoria. Se utilizaron diseños previos de Oliver [9] y Bishop [10] para la lógica de interfaz PCI, permitiendo transferencias entre el host (PC) y los bancos de memoria.

Uno de los chips FPGA restantes, se utilizó para sintetizar la circuitería de la red neuronal. También se usaron dos bancos de memoria, permitiendo así la escritura de los resultados correspondientes al set de entradas actual, al mismo tiempo que los valores de entrada del siguiente set es leído.

Dada la plataforma hardware, quedan impuestas diferentes restricciones debido a las capacidades finitas de memoria y área del dispositivo programable. Por un lado el área disponible limita el número de multiplicadores

que pueden ser sintetizados en el chip para un determinado ancho de palabra. Esta limitación puede relajarse usando repetidamente el mismo hardware como se explicó anteriormente. La memoria externa disponible impone una cota superior al número de sets de entrada que pueden ser procesadas sin intervención del host. Por otro lado, La memoria disponible on-chip limita la cantidad de pesos y bias que pueden ser almacenados, por lo tanto restringe el tamaño de la red neuronal que se puede sintetizar.

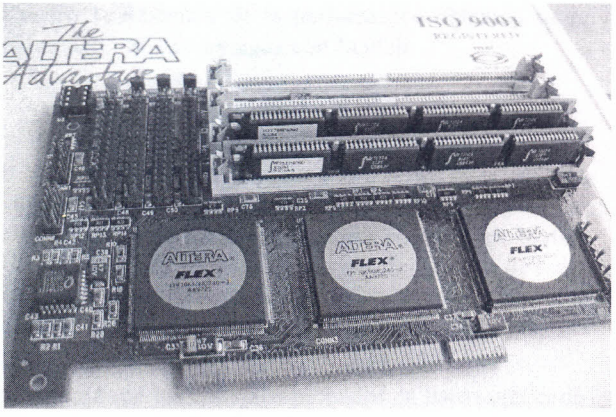

Figura 7: Placa ARC-PCI

5. Aplicaciones ejemplo

Se seleccionaron varios ejemplos de aplicación provenientes de diferentes áreas a efectos de mostrar la facilidad de uso del sistema. Una vez fijada la plataforma hardware existe un compromiso entre el ancho de palabra a utilizar (que afecta la precisión numérica de los cálculos) y la cantidad de multiplicadores que pueden acomodarse en el área de silicio disponible (y en consecuencia la velocidad de cálculo). En cada ejemplo, una vez dada la descripción de la red entrenada se realizaron varias compilaciones para seleccionar la solución más rápida con un error aceptable. Todo este proceso fue realizado en un tiempo breve, del orden de un día de trabajo para cada ejemplo, mostrando la flexibilidad del ambiente obtenido.

Se muestran los resultados para dos aplicaciones que fueron sintetizados y luego probados en la placa ARC-PCI. El primero es un problema clásico de clasificación de patrones, y el segundo es una función de aproximación. Además de las aplicaciones probadas sobre la placa ARC-PCI se sintetizaron varias redes para chips más recientes y la velocidad de cálculo resultante fue evaluada.

5.1. Problema de clasificación

La aplicación consiste en el reconocimiento de los fonemas de las cinco vocales del idioma español dado un conjunto de características extraídas de una señal de voz muestreada.

La arquitectura de la red es la siguiente: 2 capas, 8 entradas, 6 neuronas en la capa de entrada y 5 neuronas en la capa de salida, una para cada vocal. La presencia (o ausencia) de una vocal se codifica como un valor '1' (o un valor '-1') en la salida correspondiente.

El punto de partida fue una red entrenada con patrones normalizados en el rango $[-1, 1)$, usando la tangente hiperbólica como función de activación. El circuito se sintetizó para diferentes combinaciones de los parámetros. La performance obtenida fue de 490 mil juegos de entrada por segundo (alrededor de 38 millones de productos peso-entrada por segundo), usando un reloj de 16.67 MHz en un circuito con 8 multiplicadores de 8 bits. Este circuito clasificó correctamente todos los juegos de entrada probados.

El problema fue tomado de una aplicación más compleja desarrollada como proyecto final en un curso de grado (Facciolo et al. [11]).

5.2. Problema de aproximación funcional

Este problema fue obtenido de PROBEN1 (Prechelt [6]), y consiste en la predicción del consumo de energía en un edificio. El objetivo es predecir el consumo por hora de energía eléctrica, agua caliente y agua fría, como una función de la fecha, la hora del día, la temperatura y humedad ambiente, la radiación solar y la velocidad de viento. La red tiene 14 entradas, 3 capas de 8, 8 y 3 neuronas en la primera, segunda y tercera capas respectivamentee. La función de activación es omitida en la capa de salida.

La red fue entrenada en Matlab con un conjunto de 2104 ejemplos, utilizando la función de activación $\tanh(x)$.

El circuito fue sintetizado para varias combinaciones de parámetros. Con la configuración seleccionada se obtuvo una performance de 231 mil juegos de entrada por segundo, o en forma equivalente 46,3 millones de productos peso-entrada por segundo. Al igual que en el ejemplo anterior se utilizó la frecuencia de reloj de 16,67 MHz y se utilizaron también 8 multiplicadores de 8 bits. El error numérico fue menor que 0,77 por ciento del fondo de escala de las señales utilizadas.

6. Resultados

A manera de resumen, la tabla 1 muestra una comparación de performance entre el circuito sintetizado en la placa ARC-PCI y una solución "solo software" desarrollada utilizando el toolbox de redes neuronales de Matlab, para los dos ejemplos de aplicación mencionados más arriba.

Cabe aclarar que si bien Matlab utiliza una representación numérica más precisa, igual es presentada aquí como referencia dado que es una herramienta de simulación ampliamente utilizada.

	ARC-PCI	Matlab
Problema de Clasificación	490.3 Ksets/s 38.24 Mprod/seg	117 Ksets/s 9.13 Mprod/s
Aproximación Funcional	231.5 Ksets/s 46.31 Mprod/seg	43 Ksets/s 8.60 Mprod/s

Cuadro 1: Resumen de la performance obtenida sobre placa ARC-PCI comparada con Matlab

En ambos casos el circuito utiliza 8 multiplicadores de 8 bits cada uno y un reloj de 16,67 MHz, resultando en cada caso una ocupación de 85 % y 91 % del área del chip FLEX10K50. La performance obtenida en Matlab fue medida utilizando la función *sim* del toolbox de redes neuronales (con la precisión completa de 64 bits en punto flotante utilizada por Matlab), sobre Windows XP en una cpu Athlon XP 1700+ con 256MB de RAM.

Además de los ejemplos probados en hardware, el diseño fue sintetizado para chips más recientes en tecnologías APEX II, ACEX y Stratix de Altera, que son chips más grandes y rápidos que los de la placa ARC-PCI. Los mejores resultados fueron para un chip de la familia Stratix donde las herramientas de síntesis reportaron una performance esperada de **360 millones de productos por segundo** para una red grande, utilizando 48 multiplicadores de 16 bits cada uno y a una frecuencia de reloj de 35 MHz. A la fecha de finalizado el proyecto el precio de un chip de ese porte era de aproximadamente USD 1200 comprado en pequeñas cantidades.

7. Conclusiones

Se obtuvo un diseño flexible y escalable para toda una familia de redes neuronales del tipo perceptrón multicapa. En cada nueva aplicación el circuito puede obtenerse seleccionando los valores de los parámetros adecuados para las dimensiones de la red en particular y sintetizando el circuito. En forma similar, para una red determinada se pueden sintetizar facilmente diferentes combinaciones de parámetros para seleccionar la mejor solución para un hardware dado.

Los ejemplos mostraron que pueden obtenerse ciclos de diseño breves, permitiendo llegar a una aplicación funcionando en tiempos muy breves desde la concepción inicial.

El diseño fue validado en hardware sobre una placa con FPGAs. Incluso con la placa relativamente antigua que fue utilizada se obtuvo una mejora en velocidad apreciable en comparación con una solución software pura basada en el toolbox de redes neuronales de Matlab.

8. Reconocimientos

Queremos agradecer a la empresa Altera por proveer la placa ARC-PCI sobre la que este trabajo fue implementado, y a Juan Pablo Oliver de nuestro instituto por proveernos el diseño del core PCI utilizado.

Referencias

[1] Haykin, Simon, *Neural networks: a comprehensive foundation*. Second edition, Prentice-Hall 1991, ISBN 0-13-273350-1.

[2] James A. Freeman/David M. Skapura, *Neural Networks - Algorithms, Applications, and Programming Techniques*. Addison-Wesley, 1992, ISBN 0-201-51376-5.

[3] Marco A. Arroyo León, Arnoldo Ruiz Castro, Raúl R. Leal Ascencio, An Artificial Neural Network on a Field Programmable Gate Array as a virtual sensor, *Proceedings of the International Symposium on Robotics and Automation - ISRA'98*, Saltillo, Coahuila, Mexico, 1998.

[4] Cuauhtemoc Aguilera Galicia, Raúl R. Leal Ascencio, A CPLD Implementation of an Artificial Neural Network for Instrumentation Applications, *Second International Workshop on design of Mixed-Mode Integrated and Applications*, Guanajuato, México, 1998.

[5] Charles E. Cox and W. Ekkehard Blanz. GANGLION - A Fast Field Programmable Gate Array Implementation of a Connectionist Classifier, *IEEE Journal of Solid-State Circuits*, 27(3):288-299, March 1992.

[6] Lutz Prechelt, *Proben1 - A Set of Neural Network Benchmark Problems and Benchmarking Rules*. Fakultät für Informatik, Universität Karlsruhe, Germany, 1994 Technical Report.

[7] D. Hammerstrom, *A VLSI Architecture for High-Performance, Low-Cost, On-chip Learning*. Proceedings of the International Joint Conference on Neural Networks. 1990

[8] B. Girau, A. Tisserand, *On-line Arithmetic based Reprogrammable Hardware Implementation of Multilayer Perceptron Back-Propagation*. IEEE Computer Society, 5th International Conference on Microelectronics for Neural Networks and Fuzzy Systems (MicroNeuro96). 1996

[9] Juan Pablo Oliver, Algoritmos en lógica programable, *Tesis de maestría en curso*, http://iie.fing.edu.uy/~jpo

[10] William Bishop, The ARC-PCI board page, http://www.pads.uwaterloo.ca/~wdbishop/arc-pci/

[11] Gabriele Facciolo/Diego Rother, *Reconocimiento de Voz*. Proyecto final para el curso "Sistemas Neuro-Fuzzy", 2002. http://iie.fing.edu.uy/ense/asign/neurofuzzy/

[12] D. Ferrer/R. González/R. Fleitas/J. Pérez/R. Canetti, *NeuroFPGA - Implementing Artificial Neural Networks on Programmable Logic Devices*. Design, Automation and Test in Europe 2004 (DATE'04), Paris, France, February 16-20, 2004.