RIELAC, Vol.XXXVI 1/2015 p.95-106 Enero - Abril ISSN: 1815-5928

RIELAC
Revista de Ingeniería Electrónica, Automática y Comunicaciones

Implementación mediante hardware de una Red Neuronal Artificial para Reconocimiento de Caracteres

Franky Polanco Carrillo, Daniel Álvarez Gerónimo, Valery Moreno Vega

RESUMEN

En el presente trabajo se abordan temas de interés relacionados con la implementación de Redes Neuronales Artificiales (RNAs) en hardware reconfigurable. Para alcanzar dicho fin, se desarrolló una aplicación para el Reconocimiento de Caracteres, utilizando en el proceso de clasificación una red de tipo Perceptrón Multicapa (Feed-Forward Backpropagation). Los primeros pasos realizados consistieron en la creación, entrenamiento y simulación de dicha red. Para esto, se empleó la Interfaz Gráfica de Usuario (IGU) que ofrece el toolbox de Redes Neuronales de Matlab. La implementación hardware de la RNA, se realizó mediante la traducción del modelo computacional a un modelo sintetizable en hardware, el cual es descrito empleando un flujo de diseño basado en modelos, que se apoya en el entorno Matlab/Simulink y la herramienta System Generator de Xilinx. La implementación física se llevó a cabo empleando la tarjeta de desarrollo Atlys de la compañía Digilent. El trabajo está dirigido al análisis de los principales aspectos a tener en cuenta a la hora de llevar a cabo la implementación hardware de una RNA en FPGA y la metodología a seguir para alcanzar dicho fin, y que constituyen las principales novedades de este trabajo.

Palabras claves: Red Neuronal Artificial, Reconocimiento de Caracteres, FPGA, Xilinx System Generator.

ABSTRACT

This paper deals with the implementation of an Artificial Neural Network on reconfigurable hardware. The authors developed an application for character recognition using a feedforward backpropagation neural network, developed on an FPGA. In order to carry out such implementation the computational model of the neural network is translated into a FPGA synthesizable model using the Matlab/Simulink environment as well as the System Generator Xilinx tool. The results are illustrated in a prototype developed on an Atlys board. The authors focus the analysis and discussion on the main aspects considered to synthetize an ANN on an FPGA as well as the methodology to follow during such implementation, which are the main contributions of this paper.

Key words: *Artificial Neural Network, Characters Recognition, FPGA, Xilinx System Generator*

INTRODUCCION

El desarrollo de las RNA desde los años 80 del siglo pasado y su utilización en la solución de muchos problemas prácticos ha sido, al menos hasta hace pocos años, mediante implementaciones software de los modelos de RNA. Este hecho se debe a dos razones fundamentales: la dificultad de diseñar y sintetizar en hardware modelos de neuronas interconectados (que pueden ser cientos o miles) debido a que involucra aritmética de punto fijo y/o flotante y a los tiempos de procesamiento y paralelismo inherente a las RNA, sobre todo cuando el procesamiento debe ser en tiempo real. Por su parte, la implementación software, sacrifica el paralelismo pero permite una buena representación numérica de los parámetros de la red y un tiempo de procesamiento aceptable en las PC actuales.

En los últimos años sin embargo, los avances alcanzados en la implementación de RNAs mediante hardware están ligados al desarrollo de la tecnología microelectrónica de alta escala de integración (VLSI) y al caso particular de los FPGAs [1,2]. Estos avances facilitaron la síntesis hardware de algunos modelos de RNAs de alto nivel de complejidad, permitiendo su aplicación en problemas prácticos en los que alcanzaron desempeños comparables con la implementación software del mismo modelo de RNA. Con los nuevos desarrollos se incorpora un alto grado de paralelismo en el procesamiento de la RNA y una reducción considerable del costo de la aplicación al no requerir de una PC [3]. Se destaca sin embargo, que aún cuando se implementen mediante hardware, la simulación software siempre será el primer paso en la implementación y es de gran ayuda para la prueba de diseños.

En este trabajo se aborda la implementación de una RNA para reconocimiento de caracteres en hardware reconfigurable. El problema de reconocimiento de caracteres, mediante software, es conocido y abordado con cierto éxito (el error de reconocimiento es bajo pero no es cero) [4], pero no se ha abordado de la misma manera a nivel de implementaciones hardware [5,6]. La razón fundamental sigue siendo que muchos modelos de RNA trabajan con parámetros que son de "naturaleza analógica", en el sentido de que las señales que procesan no son binarias o bipolares si no que se codifican como números reales, o valores analógicos [2],[6], lo que ha contribuido a que al evaluar su implementación hardware en las últimas décadas se valora el uso de circuitos analógicos por encima de los digitales. No obstante, se pueden encontrar trabajos ([1], [3], [5], [6], [11]) donde se utilizan dispositivos FPGA para implementar mediante hardware algunos modelos y aplicaciones de RNA. Al implementar estos modelos con este tipo de dispositivos se debe considerar la representación del número real con formato de punto fijo (y no aritmética flotante) por lo que aspectos tales como la precisión y representación de los datos con este formato cobran mayor importancia. También importante es el consumo de recursos del dispositivo que conlleva tanto la RNA como este tipo de formato, ya que por lo general una RNA está constituida por decenas de neuronas densamente conectadas, y la descripción de esa arquitectura mediante hardware se expresa en una gran cantidad de recursos del FPGA. A pesar de que estos aspectos constituyen elementos importantes a considerar, y que pueden incluso hacer inviable la implementación de algunas arquitecturas de RNA, la velocidad de procesamiento, inherentemente paralela de una RNA, justifica su estudio.

El objetivo fundamental es entonces demostrar la probabilidad de llevar a cabo la implementación hardware de una RNA para reconocimiento de caracteres en hardware reconfigurable haciendo uso de la tecnología actual, evaluando para ello aspectos tales como precisión, velocidad del procesamiento paralelo que se realiza y la reducción de costo que presenta si se compara con su implementación software. Se utiliza un modelo de red conocido como Perceptrón Multicapa (PM) que se sintetiza en un FPGA de Xilinx. Como objetivo secundario se documenta la metodología de diseño empleada, así como detalles de manejo de las herramientas utilizadas, para que sirva de referencia a aquellos investigadores que aborden otros tipos de aplicaciones de RNA sintetizadas mediante hardware.

Breve descripción de la RNA utilizada en el reconocimiento de caracteres.

La red utilizada para el reconocimiento de caracteres es una de las configuraciones típicas aplicadas a problemas de clasificación de patrones, puesto que el reconocimiento de caracteres, es un caso particular de clasificación. En este tipo de aplicaciones cada carácter se asocia a una clase (la cantidad de clases depende del alfabeto que se utilice). La tarea de la RNA es clasificar correctamente cada entrada (carácter) que se le presente en la correspondiente clase codificada a la salida de la red. La red Perceptrón Multicapa es la arquitectura por excelencia empleada en este tipo de problemas, utilizándose en más del 70% de las aplicaciones [7,8]. La figura 1 muestra el esquema de la RNA seleccionada (ver [3] para detalles de cómo se llega a esta arquitectura). La capa de entrada cuenta con 35 neuronas (la matriz del carácter es de 7x5), la capa oculta contiene 16 neuronas y la capa de salida 7 (la clase la codifica de acuerdo al ASCII de la letra).

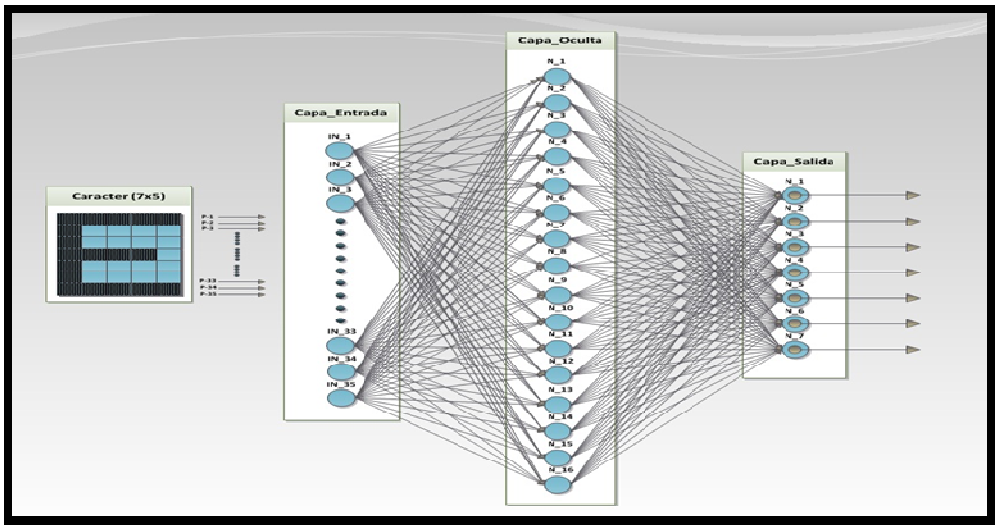


Figura 1 Esquema de la red Perceptrón Multicapa empleada en la aplicación

Los patrones de entrada empleados en el entrenamiento están formados por vectores columna. La dimensión de estos es de 35x1, donde la posición (1,1) está ocupada por el bit más significativo (MSB) del carácter y la posición (35,1) por el menos significativo (LSB). La asociación de los bits que forman el carácter y el vector, se realiza llevando las filas a columnas de manera que los cinco bits de la primera fila del carácter ocupan las cinco primeras posiciones del vector columna, los cinco bits de la segunda fila se ubican a continuación y así sucesivamente. Los patrones de salida empleados en el entrenamiento poseen una dimensión de 7x1, buscando tener a la salida de la red el código ASCII del carácter a reconocer. Se emplean 7 bits en los patrones de salida, ya que el MSB del código de los caracteres ASCII a representar (todos letras y números) es cero para todos los patrones, o sea, siempre menor que 128 (2^7). En la implementación hardware se agrega el MSB siempre en cero para conformar el byte completo del código ASCII. El entrenamiento utilizado es el usual en este tipo de aplicaciones [3]. La red debe entrenarse para cada carácter a reconocer. En este trabajo se utilizaron 36 patrones para cada uno de los caracteres que forman el alfabeto de la aplicación. En la figura 2 se muestra un ejemplo de los utilizados para la letra E.

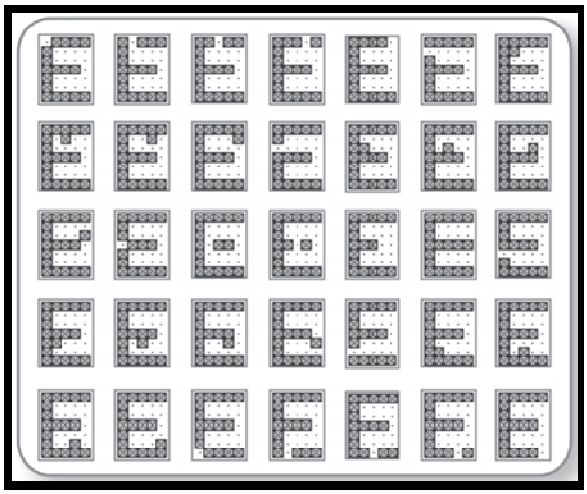


Figura 2 Patrones generados para el carácter "E", con un 2.86% de ruido

Metodología para la implementación de la RNA

La metodología a seguir en la implementación de la red Perceptrón Multicapa para el Reconocimiento de Caracteres se observa en la figura 3. Los tres primeros pasos consisten en crear, entrenar y simular la red. Para alcanzar este fin se empleó la Interfaz Gráfica de Usuario (IGU) que ofrece la biblioteca de funciones (nnet) de Redes Neuronales de Matlab, realizando la parte netamente software del proceso, en la cual se debe seleccionar el tipo de red neuronal y su configuración de capas y neuronas. Posteriormente la red es entrenada y simulada de forma iterativa hasta que se consiga la caracterización deseada de los patrones de simulación. Estos tres pasos de la metodología de diseño son los clásicos utilizados para implementaciones software de RNA (ver [1],[3],[7],[10],[12],[13] y las referencias contenidas en dichas fuentes), y tienen como finalidad ajustar los parámetros internos de la RNA para su posterior utilización.

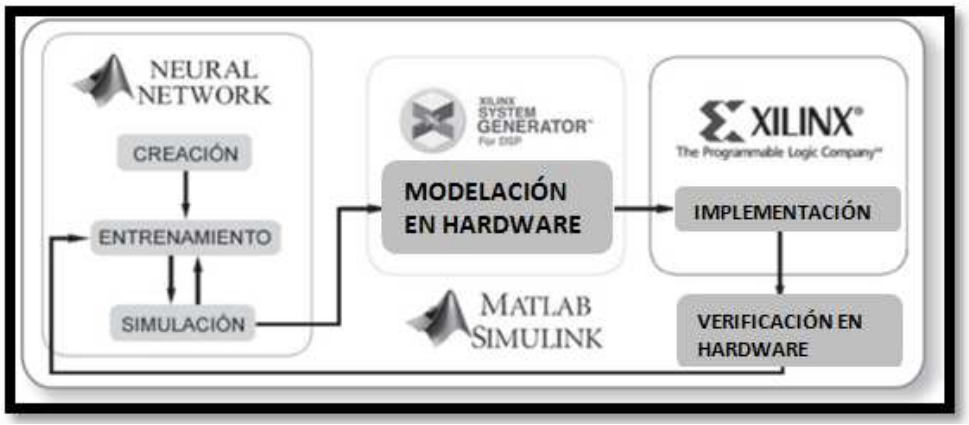


Figura 3 Metodología para la implementación de la red Perceptrón Multicapa

Una vez cumplidos los 3 primeros pasos, se pasa a la etapa de modelación en hardware. Para esto se emplea la herramienta Xilinx System Generator (XSG), que utiliza Simulink como entorno de desarrollo. La modelación en hardware consiste en crear el diseño de la red empleando para esto los bloques proporcionados por XSG. Por último se pasa a la etapa de implementación y verificación de hardware. Para alcanzar este fin, XSG permite realizar una Co-Simulación, que consiste en simular el diseño desde el FPGA e integrarlo directamente en el espacio de trabajo de Simulink. Las siguientes secciones desarrollan cada uno de los pasos de la metodología relacionados con la síntesis por hardware de la RNA en el FPGA.

Modelación en Hardware de la RNA.

La herramienta XSG es una de las utilizadas en la actualidad en aplicaciones relacionadas con la implementación de funciones en dispositivos lógicos programables, debido al alto grado de abstracción que presentan los diseños basados en bloques, que permiten una mejor comprensión de los mismos y una considerable reducción de los tiempos de implementación y puesta a punto de una determinada aplicación. En este trabajo además se seleccionó dicha herramienta para realizar la síntesis de la RNA en hardware, pues permite el desarrollo e implementación del sistema empotrado sin necesidad de conocer a fondo la programación en HDL, realizando los procesos de síntesis, implementación y descarga en la placa automáticamente desde un modelo en Simulink [9]. También se destaca la posibilidad de simulación del diseño desarrollado en software, con MATLAB/Simulink, y su verificación en hardware utilizando Co-Simulación [9]. Debe, sin embargo, garantizarse la adecuada parametrización de los modelos para brindar una solución viable a las necesidades planteadas y a la utilización de los recursos del dispositivo lógico programable.

La unidad básica de procesamiento de información en la RNA es la neurona artificial. El modelo matemático ([10]) que la representa es:

salida*=∑(w_i*x_i)+*b

donde:

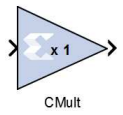
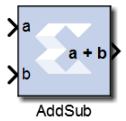
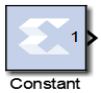
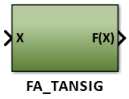
$\mathbf{w_i}$ representan los pesos de las conexiones

$\mathbf{x_i}$ representa la entrada i (***i=1..n***)

b el bias o neurona independiente.

Para realizar el diseño y posterior implementación de una neurona, se emplearon los bloques mostrados en la Tabla 1. Estos bloques implementarán el modelo matemático de la neurona artificial.

Tabla 1 Elementos que forman la Neurona Artificial

Elementos que forma la Neurona	Bloques (XSG)
Multiplicador por una Constante (CMult): Este bloque se comportan como una ganancia, obteniéndose a la salida el producto de la entrada por una constante definida por el usuario. Es el equivalente a los pesos en modelo artificial de una neurona	
Sumador (AddSub): En el diseño de la neurona se encarga de sumar las salidas de los bloques CMult.	
Constante (Constant): Es similar al bloque de Simulink empleado para representar una constante, pero puede ser conectado directamente a los bloques de XSG. Su función es la de representar el Bias.	
Función de Transferencia: Es un subsistema encargado de implementar la función de transferencia tangente sigmoidal.	

De la figura 1 se aprecia que cada neurona de la capa de salida recibe la salida de 16 neuronas de la capa intermedia. Esta arquitectura, utilizando XSG y los bloques de la Tabla 1 se pueden sintetizar como se muestra en la figura 4. En dicha figura se observan 16 multiplicadores (CMult) en paralelo, que representan los enlaces sinápticos o pesos. La salida de los multiplicadores se conecta a una red de 16 sumadores (AddSub) en cascada que suman el aporte de cada enlace proveniente de los multiplicadores más el Bias. Por último, el resultado de la sumatoria es pasado al subsistema que implementa la función de activación, *FA_TANSIG*, y su salida constituye la salida de la neurona. Este diseño se repite para cada neurona de la capa de salida. El diseño realizado para las neuronas de la capa de salida cuenta con 33 bloques en total, que deben ser configurados previamente (figura 4), lo que significa parametrizar de 2 a 6 variables en el XSG por cada bloque utilizado. Este proceso de configuración podría resultar demasiado lento y retrasar la puesta a punto del diseño. Para facilitar esta tarea, se puede definir un subsistema que represente al conjunto de diseño de la neurona de salida (o sea, a sus 33 bloques) de manera que parametrizando el subsistema y replicándolo se pueda crear la capa de salida en menor tiempo y con menos probabilidad de error de configuración. La parametrización se puede realizar definiendo una máscara, que utilizando XSG significa que se le asigna al subsistema un cuadro de diálogo personalizado que permite configurar los bloques contenidos dentro del subsistema. La figura 5 muestra la máscara asociada a la Neurona de la Capa de Salida. La máscara posee en la parte superior una breve descripción del subsistema. En el centro se observan 3 secciones (tabs) que permiten configurar los boques del diseño: multiplicadores, sumadores y la neurona independiente (Bias). Además cuenta con una breve ayuda sobre la representación numérica empleada.

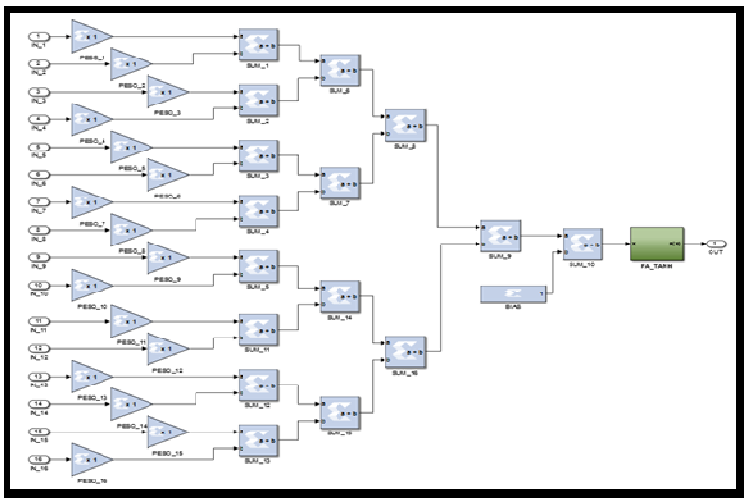


Figura 4 Diseño de una neurona de la capa de salida

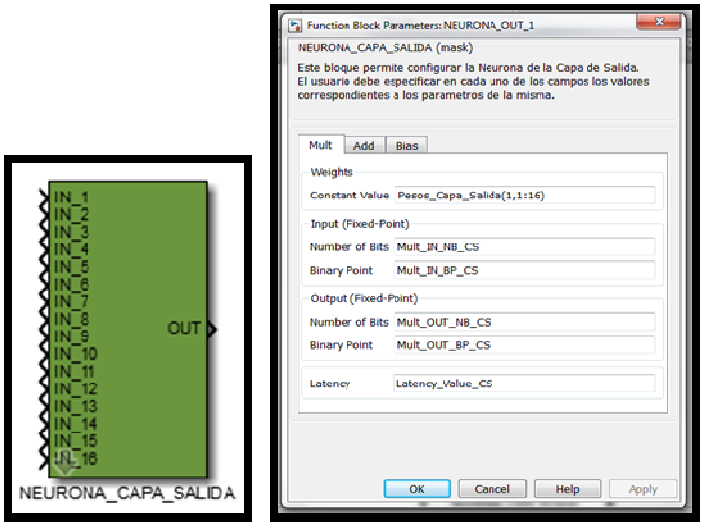


Figura 5 Subsistema y Máscara asociada al diseño de la neurona de la capa de salida

El mismo principio se puede seguir desarrollando para crear entonces un subsistema que represente la Capa de Salida (Figura 6) y que esté formado internamente por 7 subsistemas del tipo “neurona de la capa de salida”. Esta metodología de diseño facilita la comprensión del mismo por terceros, ya que parte del componente de menor complejidad (neurona) hasta el de mayor complejidad (capa de neuronas). También se utiliza el mismo principio para construir la neurona de la capa oculta, y el subsistema capa oculta (formado por los subsistemas “neurona de capa oculta”). La diferencia que presenta esta capa con la anterior, es que posee 35 entradas (caracter) y 16 salidas. Finalmente, conectando los subsistemas Capa Oculta y Capa de Salida se obtiene el diseño de la RNA completa (Figura 7).

En una RNA implementada mediante software, los parámetros de la red se representan por números reales de simple o doble precisión (representación decimal conocida por “coma flotante”). Este tipo de datos es complejo de implementar en un FPGA, por lo que se prefiere siempre la representación decimal de punto fijo. El problema de la representación de punto fijo es determinar la cantidad de bits que garantizan la precisión requerida por diseño y no más de los necesarios, pues se estarían consumiendo recursos adicionales del FPGA ([14], [16]). La forma tradicional de realizar este balance entre precisión y consumo de recursos es mediante la realización de sucesivas simulaciones de la red, variando el número de bits empleado en cada uno de los bloques.

Las máscaras creadas (ver figura 7), ayudan a que los cambios de configuración de la red neuronal se realicen de forma sencilla y rápida.

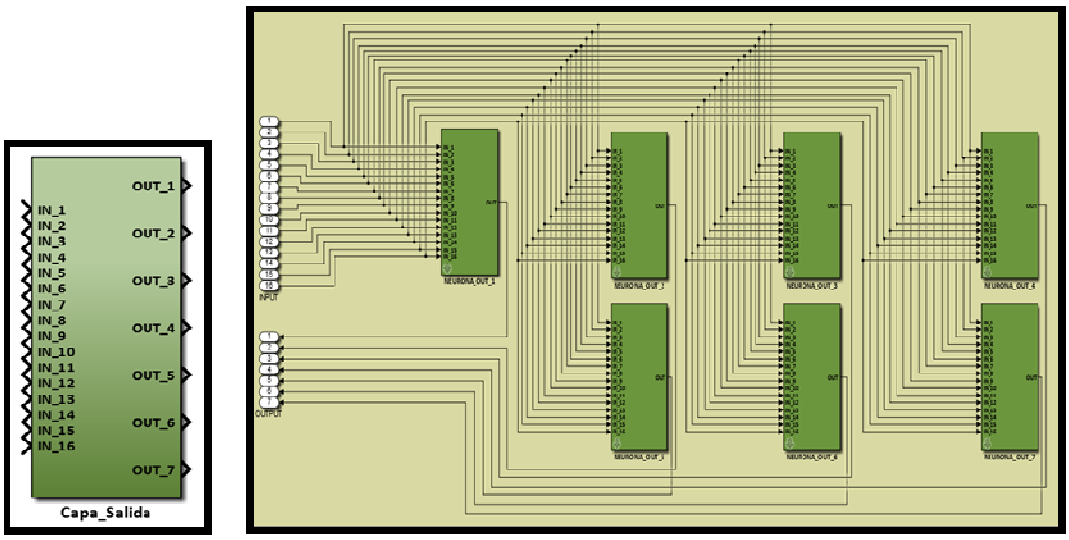


Figura 6 Diseño de la Capa de Salida

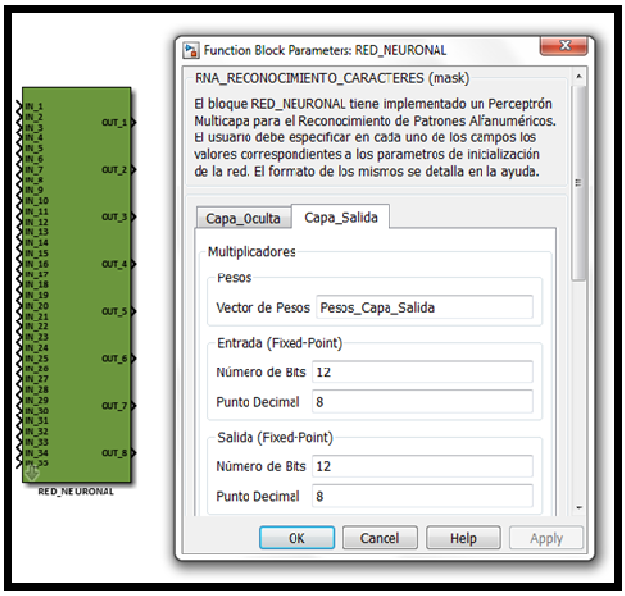


Figura 7 Subsistema y Máscara asociada a la RNA

Otro aspecto de interés es la implementación de la función de activación asociada a cada una de las neuronas de la RNA. Esta función es la tansig [11] que tiene secciones lineales y no lineales. En el presente trabajo, se emplean para el diseño de la función de transferencia de cada neurona la combinación de los dos principales métodos de implementación de funciones mediante hardware: Aproximaciones Lineales y Tablas de Búsqueda [3,12], razón por la cual se le denomina método híbrido, y es uno de los aportes realizados. Las tablas de búsqueda necesitan, para su implementación, de bloques de RAM en el FPGA lo que aumenta el consumo de recursos. Por otra parte, las aproximaciones lineales consumen pocos recursos (a lo sumo se necesitarían un sumador y un multiplicador por una constante) pero no son exactas si la función no es lineal. El método propuesto emplea la Tabla de Búsqueda en los tramos donde el comportamiento de la función es no lineal y el de aproximaciones lineales en los

tramos donde la función tiene un comportamiento lineal. Esta combinación garantiza un equilibrio entre precisión y consumo de recursos. En la próxima sección, al abordar aspectos relativos a la síntesis del diseño, se analiza el error para la implementación seleccionada.

En la figura 8 se observa el diseño de la función de transferencia tangente sigmoidal empleando los bloques de XSG. El bloque selector (MCode) analiza el valor de entrada "X" y genera una salida de selección asociada a este valor. Por ejemplo, para valores de "X" comprendidos entre [-0,5 0,5], el comportamiento de la función "tansig" responde a la recta y = x, por lo que la salida de selección será "000". Para el rango de valores entre (0.5 3], el comportamiento de la función es no lineal, en este caso se emplea una tabla de búsqueda, la salida de selección será "001". Para los valores comprendidos de [3, +∞), la función "tansig" se satura en uno, en este caso la salida de selección será "010". Para el rango de valores negativos se procede de la misma manera, teniendo en cuenta que la función tangente sigmoidal es impar. La demora introducida entre el bloque selector y el multiplexor se emplea por cuestiones de sincronización, por la latencia asociada al bloque ROM de la tabla de búsqueda.

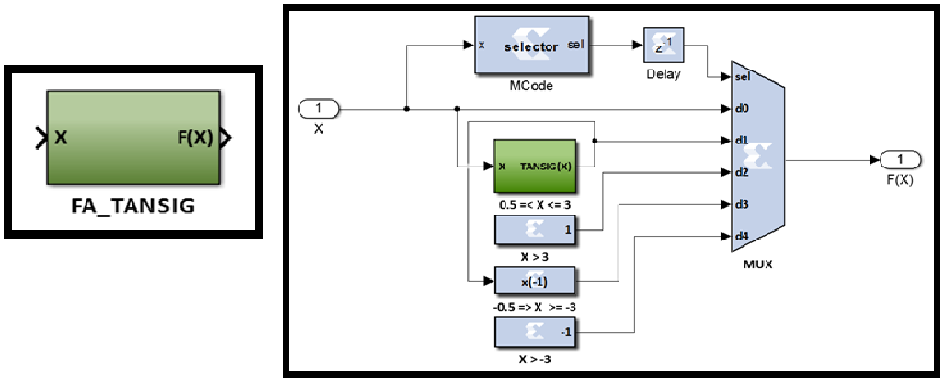


Figura 8 Función de Transferencia empleando el método híbrido

Otro aspecto muy importante a tener en cuenta cuando se trabaja con RNA es el pre y pos procesamiento de los patrones de entrenamiento, que son normalizados antes de ser aplicados a la red [3]. La normalización es importante porque las funciones de activación de las RNA están acotadas, y trabajar en la zona de saturación de estas funciones trae consigo un pobre aprendizaje de la red. Por defecto el rango de valores aceptado es de [-1 1]. Si los datos de entrada no se encuentran en este intervalo se normaliza la entrada mediante una función de pre procesamiento, y posteriormente se pos procesa la salida de la red para volver a expresar la salida en el rango de valores originales empleados en el entrenamiento.

En la síntesis hardware de la RNA que se aborda en el trabajo también se hace necesario el pre y pos procesamiento. En este caso, se implementa la función "*mapminmax*", cuya expresión es:

$$y = (y_{max} - y_{min}) * \frac{x - x_{min}}{x_{max} - x_{min}} + y_{min}$$

Para la aplicación el rango a normalizar es, para la entrada, de [-1,1] y para la salida de [0,1] debido a que la salida codifica el ASCII del carácter reconocido. En el caso del Posprocesamiento se llevará la salida de la red, del rango de valores de [-1 1] a rango de [0 1]. Después de sustituir los valores extremos en la ecuación que represente el algoritmo "*mapminmax*", se obtiene la ecuación ***y = 0.5x +0.5***.

Para el diseño del bloque de pre procesamiento se opera de la misma manera. En este caso los valores de las variables se invierten, ya que se pretende llevar del rango de [0 1] al rango de valores de [-1 1], por lo que bastaría con despejar la ecuación obtenida en el caso anterior e invertir las variables.

La figura 9 muestra el bloque de pos procesamiento diseñado. Con este bloque se garantiza una salida en el rango deseado de 0 a 1, pero no los valores 0 o 1. Es por ello que se debe además agregar un bloque Limitador Binario que reciba la salida del bloque de pos procesamiento en el rango de [0 1] con formato punto fijo, y limite la salida de la red neuronal a solo dos valores "0" o "1". Este bloque además convierte los valores de entrada de la representación en punto fijo (16 bits) a la representación booleana (1 bit). En la figura 10 se muestra el subsistema el diseño del bloque.

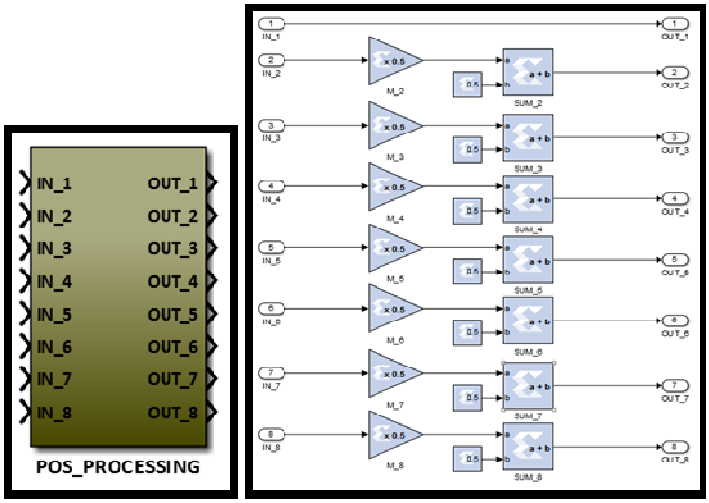


Figura 9 Diseño del bloque de Posprocesamiento

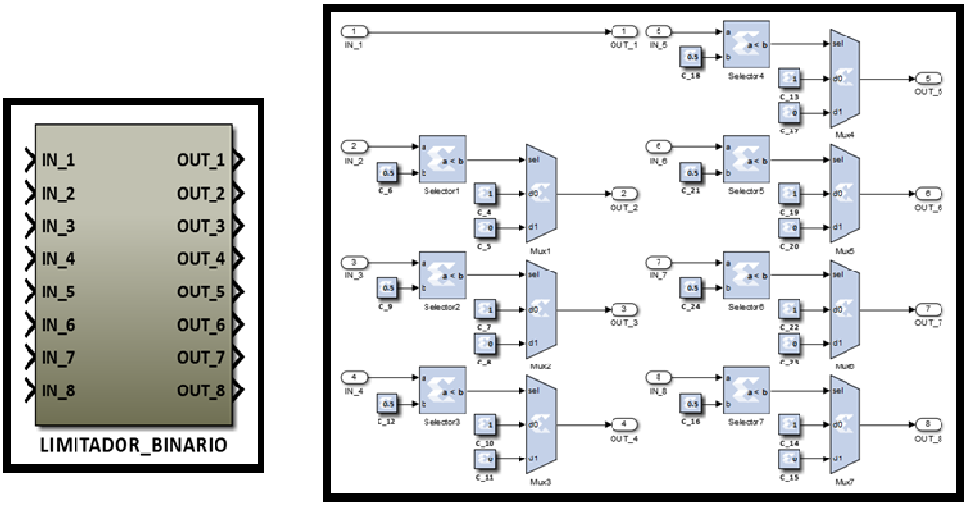


Figura 10 Diseño del bloque Limitador Binario

Implementación y verificación en Hardware

Una vez diseñado el hardware donde se implementa la RNA se procede a la simulación de este diseño antes de su implementación y verificación final en el dispositivo. Como patrón de comparación se utilizó una RNA implementada en Matlab (software). En esta etapa se puede verificar la precisión de los resultados que reporta el diseño en hardware y determinar la precisión adecuada para los bloques que trabajan con el formato de punto fijo. También en esta etapa se verifica el diseño del bloque *función de transferencia* (FA_TANSIG) por el método híbrido y se evalúa su precisión con relación a la implementación software. En la tabla 2 se muestran los resultados alcanzados al presentar el patrón 29 del carácter B a la entrada de la RNA y la salida que se obtiene para la implementación software (izquierda de la tabla) y la implementación hardware utilizando el XSG (derecha). Los campos de la tabla muestran los valores obtenidos a la salida de la capa oculta (Sal_CO), a la salida de la capa de salida (Sal_CS), a la salida de los bloques de posprocesamiento (Sal_PP) y por último el código ASCII del carácter clasificado (Sal_ASCII). Se puede comprobar la precisión de los resultados obtenidos (tres lugares decimales). Esta precisión también se mantiene a la salida de la Función de Transferencia diseñada por el método híbrido. A modo de ejemplo se resaltan algunos valores en la tabla.

La realización de la simulación también permitió analizar aspectos temporales asociados al diseño y la relación precisión-número de bits. La única demora asociada al diseño la introduce el bloque de la Función de Transferencia, producto de la latencia inherente a los bloques ROM de XSG, el resto de los bloques se ejecutarán en paralelo y de acuerdo al reloj seleccionado para el FPGA, por lo que la ejecución de la RNA implementada mediante hardware tendrá una latencia reducida. En cuanto a la precisión, se utilizó el criterio de prueba y error para determinar el número óptimo de bits a emplear en el diseño. Como punto de partida se comenzó con un total de 8 bits, de los cuales 6 representan los dígitos decimales ($Q_{8.6}$), incrementando este número de dos en dos. Los mejores resultados se alcanzaron empleando 16 bits en la representación de los datos ($Q_{16.14}$). Utilizar una representación mayor a 16 bits mantiene la precisión en tres lugares decimales pero incrementa el consumo de recursos por lo que no se recomienda. Los resultados de la tabla 2 utilizan datos ($Q_{16.14}$), mientras que en la tabla 3 se observan los resultados empleando 10 bits en la representación de los datos. Si se comparan estos resultados con los obtenidos en la tabla 2, se puede llegar a la conclusión que la representación numérica, como se espera, es importante, pero debe estudiarse para cada problema la relación de compromiso entre número de bits y precisión deseada (2, 3, o más lugares decimales) para evitar un consumo excesivo e innecesario de recursos hardware en el FPGA. En la tabla 3, a modo de ejemplo, se señala una salida de capa oculta afectada por el número de bits seleccionado utilizando la representación de los datos ($Q_{10.8}$).

Resultados de la Simulación de la Red Perceptrón Multicapa (16-7)								
Caracter	Test_RNA_Software				RNA_OCR_XSG (Simulink)			
	Sal_CO	Sal_CS	Sal_PP	Sal_ASCII	Sal_CO	Sal_CS	Sal_PP	Sal_ASCII
P_B(29)	-0.9998	--	0.0000	0.0000	-1.0000	--	0.0000	0.0000
	-0.9856	0.9998	0.9999	1.0000	-0.9855	1.0000	1.0000	1.0000
	1.0000	-0.9993	0.0003	0.0000	1.0000	-1.0000	0.0000	0.0000
	-0.9966	-0.9868	0.0066	0.0000	-1.0000	-0.9869	0.0065	0.0000
	0.9999	-0.9990	0.0005	0.0000	1.0000	-1.0000	0.0000	0.0000
	0.9992	-0.9807	0.0097	0.0000	1.0000	-0.9809	0.0093	0.0000
	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
	0.9999	-0.9997	0.0001	0.0000	1.0000	-1.0000	0.0000	0.0000
	-1.0000				-1.0000			
	0.9971				1.0000			
	-0.9982				-1.0000			
	-0.9961				-1.0000			
	0.9978				1.0000			
	-1.0000				-1.0000			
	0.8939				0.8937			
	0.9976				1.0000			

Tabla 2 Resultado de la Simulación de la Red Perceptrón Multicapa para el carácter "B" (Q16.14)

Resultados de la Simulación de la Red Perceptrón Multicapa (16-7)								
Caracter	Test_RNA_Software				RNA_OCR_XSG (Simulink)			
	Sal_CO	Sal_CS	Sal_PP	Sal_ASCII	Sal_CO	Sal_CS	Sal_PP	Sal_ASCII
P_B(29)	-0.9998	--	0.0000	0.0000	-0.9574	--	0.0000	0.0000
	-0.9856	0.9998	0.9999	1.0000	-0.9641	0.5226	0.7913	1.0000
	1.0000	-0.9993	0.0003	0.0000	0.9641	-0.6527	0.1736	0.0000
	-0.9966	-0.9868	0.0066	0.0000	-0.9545	-0.5299	0.2350	0.0000
	0.9999	-0.9990	0.0005	0.0000	0.8854	-0.3164	0.3418	0.0000
	0.9992	-0.9807	0.0097	0.0000	0.9641	-0.5980	0.2010	0.0000
	1.0000	1.0000	1.0000	1.0000	0.9641	-0.5154	0.2423	1.0000
	0.9999	-0.9997	0.0001	0.0000	0.9641	0.4930	0.7465	1.0000
	-1.0000				-0.9641			
	0.9971				0.9641			
	-0.9982				-0.9641			
	-0.9961				-0.6962			
	0.9978				0.9641			
	-1.0000				-0.9641			
	0.8939				0.9366			
	0.9976				0.0468			

Tabla 3 Resultado de la Simulación de la Red Perceptrón Multicapa para el carácter "B" (Q10.8)

En cuanto al error de aproximación máximo de la función de activación utilizando el método híbrido, se puede estimar como muy bajo. Para el caso de la representación $Q_{16.14}$ será menor al 0.24%. Note que en la Tabla 2 se aprecia que el error máximo alcanzado en la capa oculta es de este orden (diferencia entre 0.9976 y 1), pero considerando dos fuentes: la aproximación por el método híbrido y el error de aproximación debido a la multiplicación del valor a la entrada de la red y el peso asociado a esta conexión. Si se toma como referencia [16], donde se implementa una función sigmoidal logarítmica utilizando diferentes métodos y representaciones de datos, se aprecia que 0.24% es muy bueno, siendo incluso menor que los errores máximos reportados en dicha referencia para los métodos probados en ese trabajo.

Después de comprobado el diseño realizado se pasa a su implementación y verificación hardware. En estas últimas etapas se utiliza una tarjeta de desarrollo Atlys de la compañía Digilent, basada en el Spartan-6 LX45 de Xilinx [15]. La síntesis en una placa de desarrollo permite contar, además de con un FPGA, con una serie de periféricos y dispositivos externos al FPGA pero ya conectados a este (por ejemplo, visualizadores LCD o 7 segmentos, interruptores, leds, puertos serie/USB/Ethernet, etc.) que normalmente se utilizan como soporte para suministrar la información de entrada al FPGA y visualizar o representar la salida que produce el diseño sintetizado en el dispositivo programable. También permite realizar mediciones de tiempo de demora del circuito diseñado de acuerdo a diferentes configuraciones del reloj a la entrada del FPGA, y evaluar la capacidad que se necesita en el FPGA para acoger el diseño realizado. Nótese que la forma en que se ha realizado el diseño no exige el uso de un único FPGA, lo que facilita la síntesis. Pudiera pensarse, en dependencia de la capacidad del FPGA con que se cuente, en la posibilidad de implementar algunos subsistemas en un FPGA y el resto en otro, que interconectados convenientemente (por ejemplo, las salidas del primer FPGA son la entrada del segundo) conforman la síntesis total de la RNA. Esta variante, en la que se necesitaría más de un FPGA, seguiría siendo más fiable y menos costosa que una implementación software, donde se requiere de una PC y no se garantiza paralelismo ni tiempos del orden de nanosegundos de ejecución de la RNA.

En este último aspecto, o sea, el temporal, se verificó que con la tarjeta de desarrollo utilizada, y para un reloj de de 50MHz el tiempo que transcurre desde que a la RNA se le presenta un carácter en la entrada hasta que lo clasifica es de 40ns (se consumen 2 ciclos de reloj). La misma aplicación, pero implementada por software con Matlab se ejecutó en el orden de los milisegundos, por lo que la variante hardware implementada se ejecutó en el orden 10^5 veces más rápida. Se hace notar que la frecuencia utilizada no es la máxima admitida; de acuerdo al diseño realizado la frecuencia máxima reportada en la síntesis es de 280.11MHz (T=3.57 ns) lo cual pudiera reducir aún más el tiempo de respuesta de la implementación hardware con relación a la implementación software.

En la Tabla 4 se muestra el consumo de recursos de una neurona artificial implementada por hardware. Teniendo en cuenta que la cantidad total de neuronas para la arquitectura implementada fue de 23, se requerirán 804 IOBs en un solo FPGA (algunos núcleos de la familia Virtex 5 de Xilinx tienen hasta 1200 IOBs) o en varios conectados en paralelo. Por ejemplo, si se emplea un XC3S2000-5FGG676C (489 Bonded IOBs), cuyo precio aproximado es de 30 USD, y teniendo en cuenta que se necesitarían 2 dispositivos para la implementación completa de la red, el costo sería, en términos del FPGA, de 60 USD. En este tipo de aplicaciones, el resto del hardware accesorio al FPGA no excede los 40 USD por lo que el costo total es mucho menor que el de una computadora, necesaria para la implementación software.

Recursos	**Usados**	**Disponibles**	**%**
Slice Registers	8	54576	1
Slice LUTS	505	27288	1
Occupied Slices	180	6822	2
RAM 8Bs	1	232	1
BUFGMUX	1	16	6
MUXCY	424	13644	3
Bonded IOBs	35	218	16

Tabla 4 Consumo de recursos de una neurona en el diseño implementado (datos Q16.14)

CONCLUSIONES

En el trabajo se demuestra la viabilidad de la implementación hardware de RNA para reconocimiento de caracteres. La implementación hardware de la red, además de ser factible desde el punto de vista de la descripción hardware de las funciones asociadas con el modelo de la neurona, y el almacenamiento de los parámetros de la RNA utilizando un formato de punto fijo que garantiza la precisión fijada en el diseño, permite realizar el reconocimiento del carácter en un tiempo muy bajo (que puede ser del orden de decenas de nanosegundos). Este rendimiento se debe a la poca latencia y al alto grado de paralelismo que se logra en el diseño y la síntesis hardware. El trabajo también demostró la validez de la implementación de la función de transferencia de cada neurona utilizando un esquema híbrido. Es además importante señalar que la utilización de la herramienta Simulink y en especial XSG ofrece un entorno de diseño sencillo y adecuado, para el diseño y síntesis de RNA en FPGA. La conformación de subsistemas, y su parametrización haciendo uso de estas herramientas permite el desarrollo efectivo del diseño, y facilita la simulación y posterior implementación de los diseños creados sobre hardware reconfigurable, pudiéndose incluso desplegar el diseño en varios dispositivos interconectados. Finalmente, el costo de la implementación hardware es reducido si se compara con la ejecución de la RNA en una computadora.

REFERENCIAS

1. **Castro Gracias, Francisco**, "*Fundamentos para la implementación de la red neuronal perceptrón multicapa mediante software*", Tesis de grado, Universidad de San Carlos de Guatemala, 2006.
2. **A. Muthuramalingam, S. Himavathi, and E. Srinivasan**, "*Neural Network Implementation Using FPGA : Issues and Application*", International Journal of Information Technology, 2007. **Vol. 4** (2): p. 86–92.
3. **Polanco Carrillo, Franky, Álvarez Gerónimo, Daniel y Moreno Vega, Valery**, "*Desarrollo de Aplicaciones basadas en Redes Neuronales Artificiales sintetizadas en hardware Reconfigurable*", tesis de grado, CUJAE, Cuba, pp. 109, 2014.
4. **Haykin Simon (Edt)**, "*Kalman Filtering and Neural Networks*", McMaster University, Hamilton, Ontario, Canada, 2001.
5. *"Design and Implementation of Neural Network in FPGA"*, Journal of Engineering and Development, 2012. **Vol. 16** (3): p. 73-90.
6. **Zhu Jihan, Sutt Peter**, "*FPGA Implementations of Neural Networks a Survey of a Decade of Progress*", Proceedings 13th International Conference, FPL 2003, Lisbon, Portugal, September 1-3, 2003.
7. **Martín del Brío, Bonifacio y Sanz Molina, Alfredo**, "*Redes Neuronales y Sistemas Difusos*", Editorial Alfaomega RA-MA, 2a Edición, España , 2001.
8. **Hu Hen Yu, Neng Hwang Jenq, "***Handbook of Neural Network Signal Processing*", CRC Press Electrical Engineering & Applied Signal Processing Series, 2001.
9. **G. S. Luis Manuel**, "*Aceleración de Algoritmos Mediante Hardware Reconfigurable Biblioteca de Procesamiento de Imágenes para System Generator*" tesis de maestría, CUJAE, 2011.
10. **Galushkin Alexander**, "*Neural Networks Theory*", Springer, 2010.
11. **Amos R. Omondi y Rajapakse C. Jagath (Edt)**, "*FPGA Implementations of Neural Networks*", Springer, 2006.
12. **Rivera Ordoñez Cesar,** "*Reconocimiento de caracteres por medio de una red neuronal artificial*", Colombia, Respuestas **Vol. 14** p. 30-39, 2009.
13. **J. Serrano, Antonio, Soria, Emilio y D. Martín, José**, "*Redes Neuronales Artificiales*", material descargado de www.uv.es/ocw, 2010.
14. **L. Oberstar, Erick** "*Fixed-Point Representation & Fractional Math*", http://www.superkits.net/whitepapers.htm.
15. "*Atlys TM Board Reference Manual.*", Digilent Inc., 2011.
16. **J.A. Caballero Hernández, M. Díaz Salazar, M. Moradillos Paz-Lago, S. Pavoni Oliver,** "*Implementación de la Función Sigmoidal Logarítmica en un FPGA*", RIELAC, Vol.XXXV 2/2014 p.35-44.

AUTORES

- Franky Polanco Carrillo, Ing. en Automática y Computación, CUJAE, La Habana, Cuba, ***frankyp@electrica.cujae.edu.cu***
- Daniel Álvarez Gerónimo, Ing. en Automática y Computación, CUJAE, La Habana, Cuba, ***geronimo@electrica.cujae.edu.cu***
- Valery Moreno Vega, Ing. en Automática y Computación, Doctor en Ciencias Técnicas, CUJAE, La Habana, Cuba, ***valery@electrica.cujae.edu.cu***