Respuestas
Año 14
No.1
Junio 2009
ISSN 0122-820X

Reconocimiento de caracteres por medio de una red neuronal artificial

Cesar Rivera Ordoñez[1], | Jhon Jairo Santiago[1], | Julián Ferreira Jaimes[2]

Recibido:
12 de septiembre
de 2007

Aceptado:
23 de abril de 2009

Resumen

En este trabajo se presenta la implementación de un sistema de reconocimiento de caracteres en una tarjeta de desarrollo FPGA de propósito general. La clasificación de los caracteres se realiza por medio de un modelo de red neuronal conocido como Feed-forward backpropagation. Se utiliza la herramienta de redes neuronales NNTool de Matlab, para crear, entrenar y simular este tipo de Red Neuronal Artificial (RNA) con cinco diferentes patrones de entrenamiento. Para realizar la implementación, estas RNAs, son traducidas del modelo computacional a un modelo realizable en hardware, el cual es descrito mediante bloques en Matlab/Simulink y Xilinx System Generator (XSG). El archivo de configuración bitstream, necesario para la programación del FPGA, es generado por XSG para posteriormente ser implementado con Xilinx ISE foundation en la FPGA.

Palabras Clave: *FPGA, Matlab/Simulink, reconocimiento de caracteres, red neuronal artificial, Xilinx System Generator.*

Abstract

In this project we develop a characters recognition system implemented in a general purpose FPGA card. First, The characters classification is executed by a neural network model called feed-forward backpropagation. The Matlab toobox, NNTool used for neural networks is used to create, training and simulate this kind of Artificial Neural Network (ANN) with five different patterns of training. For the implementation the ANN computer model is realizes as hardware system, which is described in a block diagram using both Matlab/ Simulink and Xilinx System Generator (XSG). Subsequently, the bitstream configuration file -necessary for the FPGA programming- is generated by XSG for implementing with Xilinx ISE foundation.

Keywords: *FPGA, Matlab/Simulink, Characters recognition, Artificial neural network, Xilinx System Generator.*

[1] Ingenieros Electrónicos, Universidad Francisco de Paula Santander. Integrantes GIAC- UFPS, cesar@ingelectronico.com, santiago@ingelectronico.com

[2] Profesor Dpto. Electricidad y Electrónica. UFPS. Integrante GIAC- UFPS, jfereir@bari.ufps.edu.co

Respuestas
Año 14
No. 1
Junio 2009
ISSN 0122-820X

Introducción

El tema de las RNAs es un campo bastante amplio e interesante, además de tener muchas aplicaciones como el reconocimiento de patrones, procesamiento de señales, optimización, predicción, procesamiento de imágenes, etc [1-3]. "Los avances que se han conseguido en los últimos años en este campo han sido notorios, permitiendo soluciones más eficaces a problemas que antes eran casi imposibles de conseguir con los tradicionales sistemas de computación que trabajan bajo la filosofía de los sistemas secuenciales, desarrollados por Von Neuman"[4]. Por otra parte, en las últimas décadas, ha surgido una nueva tecnología en el desarrollo de circuitos integrados que ofrece características tales como alta velocidad de procesamiento, procesamiento concurrente y diseño jerárquico. Esta tecnología ha permitido el desarrollo de los dispositivos lógicos programables entre los cuales se encuentran los dispositivos FPGA.

Las RNAs han sido en su mayoría implementadas en software. Esto trae beneficios, ya que el diseñador no necesita conocer el trabajo interno de los elementos de la red neuronal. Sin embargo, una desventaja en las aplicaciones en tiempo real de la RNAs basadas en software, es su ejecución más lenta comparada con las RNAs implementadas en hardware. "La simulación software siempre será el primer paso en la implementación y es de gran ayuda para la prueba de diseños, pero si se desea explotar la capacidad de procesamiento paralelo así como el manejo de datos en tiempo real, las RNAs deberán de implementarse a nivel hardware" [5-6]. Los recientes avances en la lógica programable, permiten la implementación de grandes RNAs sobre un único PLD. La principal razón de esto, es la miniaturización en la tecnología de fabricación de los componentes, donde la densidad de componentes electrónicos se dobla cada 18 meses, según la ley de Moore [7-8].

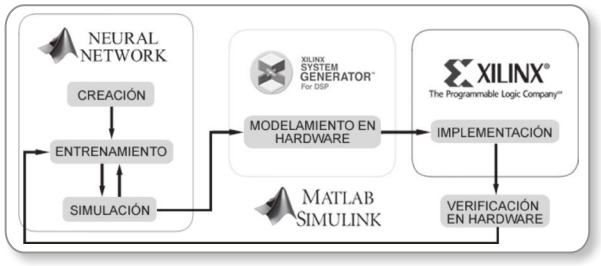
"Uno de los retos dentro del contexto de la ciencia y la tecnología, es hacer que el computador realice tareas que requieren inteligencia, razonamiento, sentido común y/o manejo del conocimiento" [9]. Ésta es una característica muy importante de una RNA, su habilidad de aprender mediante la experiencia. Cada vez que una red neuronal es creada, un entrenamiento es necesario para "enseñarle" a dar una salida, como respuesta a una entrada específica. Las redes neuronales artificiales se pueden definir como sistemas computacionales, tanto hardware como software, que imitan las habilidades computacionales de los sistemas biológicos usando un gran número de elementos simples llamados neuronas artificiales, donde su función está determinada por las conexiones entre estas. Las neuronas artificiales son emulaciones sencillas de las neuronas biológicas; toman información desde sensores u otras neuronas artificiales, realizan una operación muy sencilla con estos datos, y pasan el resultado a otras neuronas artificiales [10]. La organización de estos elementos está relacionada con la anatomía del cerebro, donde cada elemento individual se conecta a otros de forma paralela. El entrenamiento de una red para que realice una función en particular, se logra ajustando los valores de las conexiones o pesos entre sus elementos, al igual que sus puntos de ajuste o ganancias [11].

Materiales y métodos

El objetivo principal de este trabajo, es realizar la implementación de una RNA en hardware, con el fin de analizar su capacidad de responder de forma correcta a una señal de entrada con variaciones o distorsiones. Esta característica de las RNAs, de lograr dar una respuesta insensible, en cierto grado, a las variaciones de la entrada, es conocida como capacidad de generalización [12]. El proceso que se lleva a cabo para la implementación de las redes neuronales que se tratan en este proyecto se puede apreciar en la figura 1.

Respuestas
Año 14
No.1
Junio 2009
ISSN 0122-820X

Figura 1. Metodología para la implementación de redes neuronales artificiales



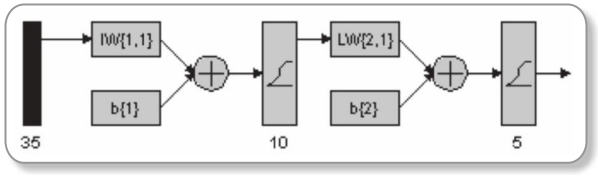
Con la herramienta NNTool de Matlab se realiza toda la parte netamente software del proceso, en la cual se debe seleccionar un tipo de red neuronal y su configuración de capas y neuronas, para posteriormente ser entrenada y simulada de forma iterativa hasta que se consigue una caracterización deseada de los patrones de simulación. Una vez se tiene entrenada la red neuronal, se crea un modelo de ésta, por medio de bloques en Simulink, para luego generar el archivo de configuración bitstream, necesario para la implementación en el FPGA.

La idea inicial del sistema de reconocimiento de caracteres consiste de una red neuronal que pudiera reconocer las letras del alfabeto cuando estas le son presentadas, incluso si se muestran de forma distorsionada. Para realizar la representación de las letras del alfabeto en hardware, se crea una matriz de interruptores de 7 filas por 5 columnas, buscando formar con este arreglo, los patrones para las letras. El estado de los interruptores (activado o desactivado), es visualizado en una matriz de 7x5 diodos LED (Diodo Emisor de Luz, del inglés Light Emitting Diode), permitiendo de esta manera visualizar la entrada a la red neuronal. Cuando un interruptor de la matriz de entrada se encuentra cerrado, este fija un estado lógico alto en la entrada de la red, lo cual se visualiza en la matriz de LEDs, con un diodo encendido en la posición correspondiente a la posición del interruptor accionado. Al igual que la entrada del sistema, la salida de este, se presenta en una matriz de 7x5 LEDs para poder evaluar la respuesta de la red.

Diseño de la red neuronal

Selección del tipo de red. Con base en los antecedentes que se tienen [1,2,4,9,12], en cuanto a la capacidad de generalización se refiere, se opta por utilizar la red neuronal feedforward backpropagation, debido a que este tipo de red es la que mejor se comporta en los problemas de reconocimiento de patrones. Para seleccionar el número de capas y el número de neuronas por capa, se procedió a la prueba y error, pasando por las tres primeras etapas de la implementación (Ver figura 1), además de tener en cuenta que la capa de salida debía constar de cinco neuronas, ya que son necesarios cinco bits para representar las 26 letras del alfabeto, excluyendo a la letra "Ñ". De esta manera se le asigna a cada letra el número que ocupa en el orden del alfabeto, de modo que a letra A le corresponde el 1, y a la Z el 26. Después de varias iteraciones y pruebas, se opta por utilizar una configuración de dos capas, 10 neuronas en la capa de entrada y 5 neuronas en la capa de salida. Algunas configuraciones de más de dos capas, arrojaron mejores resultados que los de la red seleccionada, pero no se escogió ninguna de estas porque se requerían un número mayor de neuronas, lo que a su vez supone una cantidad mayor de recursos del FPGA, a tal punto que estas redes no podrían se implementadas con los recursos disponibles. El modelo computacional de esta red creada en NNTool, aparece en la figura 2.

Figura 2. Arquitectura de la Red Neuronal Artificial implementada. Imagen tomada de NNTool



En la figura 2 se muestra que la entrada a la red neuronal consta de un vector de

Respuestas
Año 14
No. 1
Junio 2009
ISSN 0122-820X

35 elementos, correspondientes a los 35 interruptores de la matriz de entrada del sistema. Buscando limitar las salidas de las neuronas a valores entre 0 y 1, ya que este es el tipo de datos que se maneja en hardware, se usa la función de transferencia logaritmo sigmoidal. La función de aprendizaje utilizada para el entrenamiento de la red es trainscg, debido que esta utiliza uno de los mejores algoritmos de entrenamiento para problemas de reconocimiento de caracteres [13]. La función de desempeño se seleccionó a prueba y error a partir de varias etapas de entrenamiento, entregando mejores resultados la función de minimización del error medio cuadrático (MSE).

Entrenamiento. Para realizar el entrenamiento, se crearon cinco redes neuronales con las características mencionadas anteriormente. Cada una de estas se entrenó con un juego de patrones de entrada diferentes, con el fin de poder comparar y analizar las diferencias en el aprendizaje de cada una de estas y medir de esta manera su capacidad de generalización, tanto en software, así como en hardware en la etapa de validación. Para diferenciar cada una de las RNA se les asignaron los nombres RC1, RC2, RC3, RC4 y RC5. Cada una de estas redes, fue entrenada durante 1000 épocas y con una meta de error (goal) de 1x10exp-4.

Entrenamiento de la red neuronal RC1. Para esta RNA se utilizó un patrón de entrenamiento por cada letra del alfabeto, siendo este la representación ideal de la letra (sin errores), para un total de 26 patrones de entrenamiento. El mapa de bits para cada uno de estos patrones de entrenamiento, se muestra en la figura 3, donde el bit que se encuentra en la esquina superior izquierda corresponde al bit más significativo (MSB), y el bit menos significativo (LSB), en la esquina inferior derecha.

Entrenamiento de la red neuronal RC2. Para esta red se utilizaron los mismos patrones de entrenamiento de la red RC1, adicionando dos entradas distorsionadas por cada letra del alfabeto, para un total de 78 patrones de entrenamiento, tal como se muestran en la figura 4.

Entrenamiento de la red neuronal RC3. El juego de vectores de entrada para cada letra, utilizados en este entrenamiento, esta conformado por un patrón con la letra correctamente formada y dos más, con errores. En este caso, el ruido se introdujo a cada letra adicionando bits en alto, a diferencia del entrenamiento de RC2, donde se adicionaron bits en alto y en bajo en la representación ideal del carácter. La figura 5 muestra los patrones con errores introducidos en este entrenamiento.

Figura 3. Patrones de entrenamiento de la red RC1

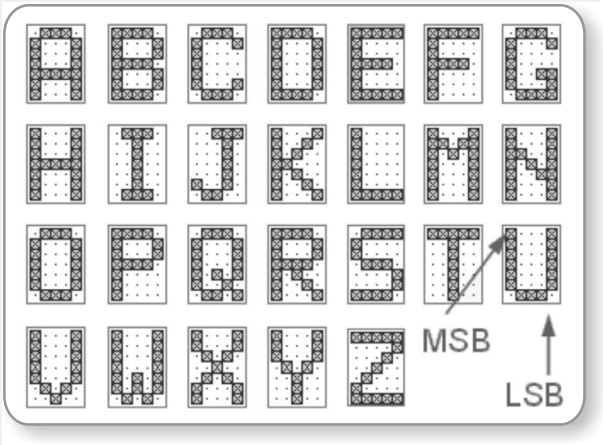
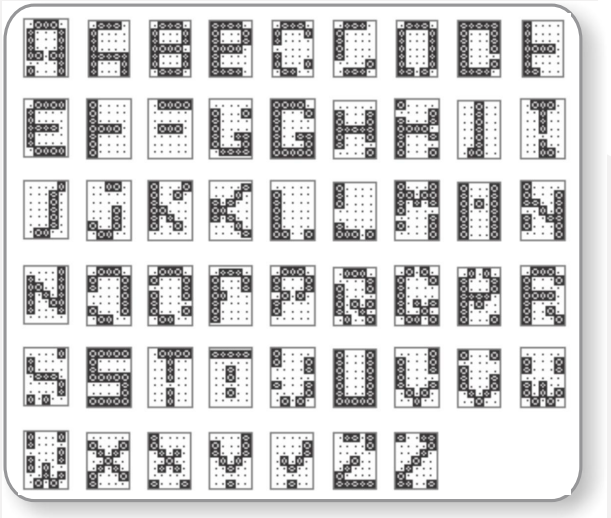
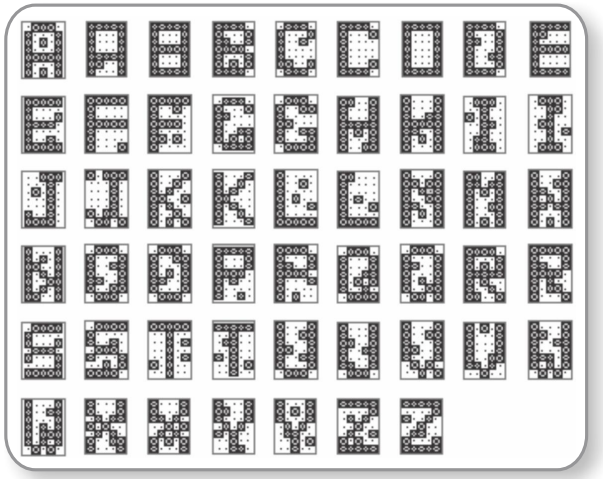


Figura 4. Patrones de entrenamiento con ruido de la red RC2



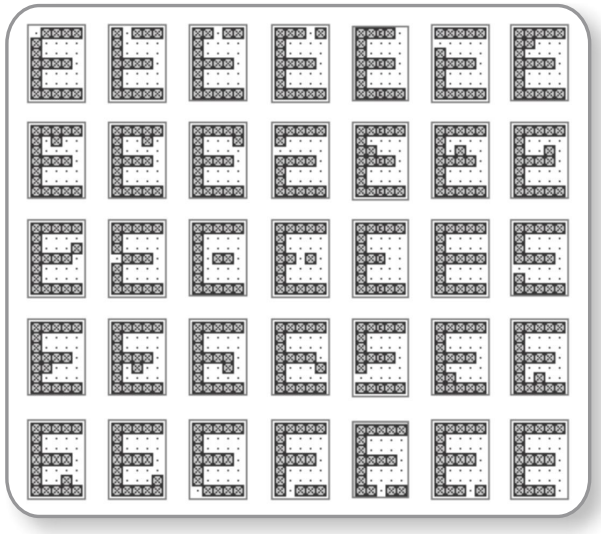
Respuestas
Año 14
No.1
Junio 2009
ISSN 0122-820X

Figura 5. Patrones de entrenamiento con ruido de la red RC3



Entrenamiento de la red neuronal RC4. En este entrenamiento se utilizaron 36 patrones por cada letra, un patrón con la letra ideal y 35 más con ruido, para un total de 936 patrones. El ruido introducido es de 2.86%, equivalente a un bit de la matriz, ya sea en bajo cuando la representación ideal de la letra tiene un determinado bit en alto, o en alto cuando el bit esta en bajo. La figura 6 muestra el caso de los 35 patrones con errores producidos para la letra E.

Figura 6. Patrones de entrenamiento con ruido para la letra E de la red RC4



Entrenamiento de la red neuronal RC5. En este entrenamiento buscando mejorar la generalización, se introdujeron al lote de 26 letras, dos caracteres más, a los cuales le fueron asignadas las salidas 27 y 28 de la red. Uno de estos caracteres es el espacio, el cual se pretende lo clasifique correctamente la red en aquellos casos donde todos o la mayoría de los bits de entrada están en cero. El otro carácter introducido en el lote de caracteres, es el inverso del carácter anterior, el cual se forma colocando todos los bits de la matriz de entrada en alto. Siendo así, en este entrenamiento se utilizaron 36 patrones por cada uno de los 28 caracteres, para un total de 1008 patrones de entrenamiento.

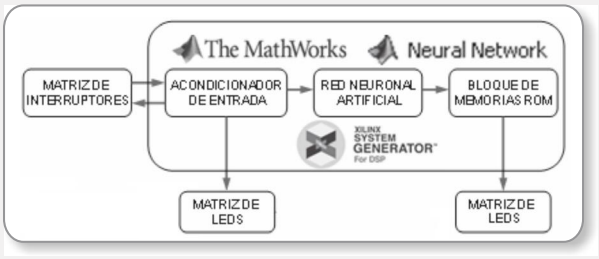
Modelamiento en XSG de la red neuronal. La estructura de la red neuronal (ver figura 2), se implementa en XSG por medio de bloques Constant (constante) para los pesos y las ganancias; bloques Mult (multiplicadores), para realizar el producto de las entradas con los pesos; bloques AddSum (sumadores), para realizar la adición de todas las entradas multiplicadas por sus respectivos pesos, y bloques ROM (memorias), las cuales se utilizan para las funciones de transferencia. El total de bloques utilizados en la red neuronal sin tener en cuenta los bloques de la función de transferencia es de 1215, a los cuales se le deben modificar 4060 parámetros para configurarlos adecuadamente. Aquí surge el inconveniente de modificar esta cantidad tan elevada de parámetros en cada una de las cinco RNAs, cada vez que se requería implementar y verificar en hardware. Para solucionar este problema, se creó un código en Matlab que permite ajustar estos 4060 parámetros de forma automática a partir de los valores para los pesos y ganancias obtenidos en el entrenamiento en NNTool, brindando una alternativa para probar en hardware las redes neuronales entrenadas.

Respuestas
Año 14
No. 1
Junio 2009
ISSN 0122-820X

Implementación

La implementación del sistema de reconocimiento de caracteres se realiza en la tarjeta de desarrollo Spartan 3 de Digilent, la cual contiene un FPGA XC3S1000 de Xilinx. Adicionalmente se diseña la tarjeta que se emplea como periférico para la Spartan 3, la cual consta de la matriz de interruptores y las dos matrices de diodos LED. Además de la RNA se deben diseñar otros subsistemas que se utilizan para acondicionar la señal de entrada y la señal de salida. El diagrama de bloques del sistema implementado se muestra en la figura 7.

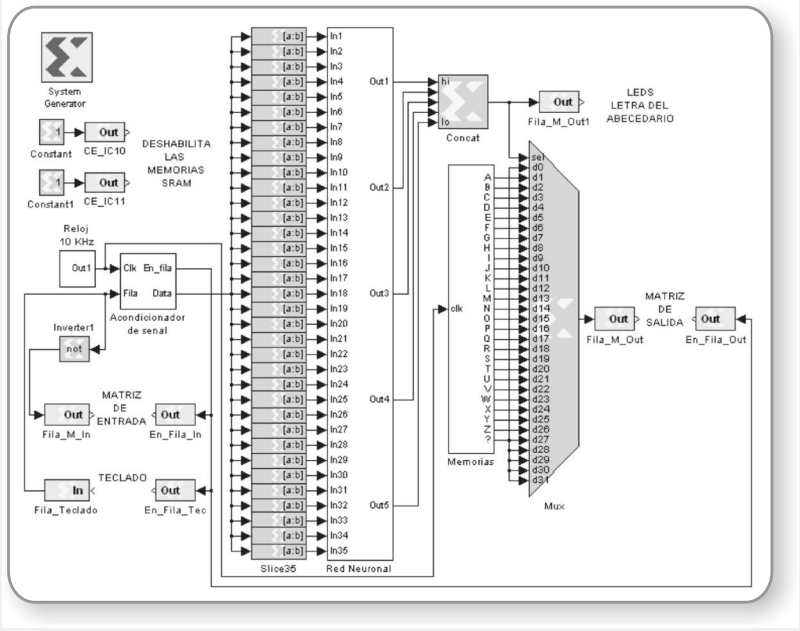
Figura 7. Diagrama de bloques del sistema de reconocimiento de patrones



Acondicionador de la señal de entrada. El objetivo de este subsistema es conformar la entrada de 35 bits de la red neuronal, a partir de la lectura secuencial de las 7 filas de 5 bits de la matriz de interruptores. Este módulo también realiza el multiplexado de las matrices de LEDs de entrada y salida. La salida Data de este bloque, corresponde al vector de 35 bits de entrada de la RNA. La frecuencia con que se realiza la lectura de la matriz de interruptores esta determinada por el reloj de entrada al bloque (Clk). La entrada fila es el vector de 5 bits que indica el estado de la fila del teclado que se está leyendo (Ver figura 8).

Bloque de memorias. La función de este bloque es guardar las matrices de bits para las letras del alfabeto y permitir la visualización de la letra que la red neuronal indique. Esto es necesario debido a que la red tiene como salida un vector de 5 bits para representar dichas letras, en lugar de una matriz de 7x5 bits, que es lo que se requiere para la visualización de la salida del sistema. Para representar en la matriz de LEDS, las salidas de la red correspondientes a los valores 27 al 31 y el 0, se crea otra matriz de bits que guarda el valor del signo de interrogación "?". Este subsistema esta conformado por bloques de memorias ROM de 5 bits de salida y 7 posiciones de memoria. El modelo en XSG del diseño implementado se muestra en la figura 8.

Figura 8. Modelo en XSG del sistema de reconocimiento de caracteres



Respuestas
Año 14
No.1
Junio 2009
ISSN 0122-820X

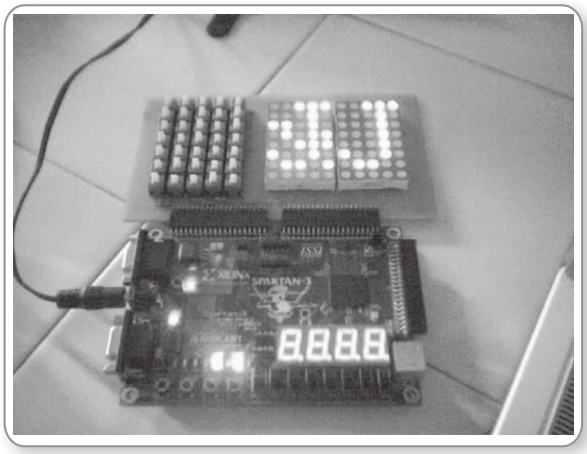
Resultados y discusión

Simulación. Para analizar el comportamiento de las cinco redes después de sus respectivos entrenamientos, se simula cada una de estas con un lote de 2626 vectores entrada-salidas esperadas, para producir un error con respecto a la salida real. Para esto, se generan de forma aleatoria, por medio de una función creada en Matlab, 101 patrones de simulación por cada carácter, con un error aleatorio que puede variar entre un rango específico, lo cual permite realizar la simulación con diferentes porcentajes de ruido en la señal de entrada.

Las entradas correspondientes a los caracteres 27 y 28 de la red neuronal RC5, no son tenidos en cuenta por esta función de Matlab. Cada una de las 5 RNAs se simuló 11 veces, variando el rango de ruido de la entrada. En la tabla 2 se muestra el resultado de la simulación para las 5 redes con porcentajes de ruido de entrada diferentes, expresado en número de bits a la izquierda y en porcentaje de error a la derecha. El desempeño de la red se mide de acuerdo con el número de entradas de la simulación, clasificadas erróneamente. En la tabla 2 se muestra este parámetro en la casilla de error (e).

Verificación en circuito. La etapa de verificación en hardware se puede realizar una vez se descarga el archivo bitstream en el FPGA. En esta etapa se observa y analiza el comportamiento de la RNA en tiempo real, ante entradas diferentes a los patrones de entrenamiento, permitiendo comparar estos resultados, con los de la simulación en software. Los datos obtenidos durante la etapa de verificación en hardware y la información contenida en la tabla 2, sirven de gran ayuda para evaluar la capacidad de generalización de las RNAs implementadas en este proyecto. La respuesta de la red RC5, ante una entrada con ruido de la letra "J" se muestra en la figura 9.

Figura 9. Verificación en Hardware



En la tabla 2 se puede apreciar que la red RC1, que fue entrenada con solo un patrón por cada carácter, tiene un mejor

Tabla 2. Resultados de la simulación expresados número de errores y porcentaje de error

Entrada # bits	RC1 e	RC2 e	RC3 e	RC4 e	RC5 e
1-2	869	980	989	328	279
1-3	1054	1170	1155	597	542
1-4	1195	1312	1310	771	720
2-3	1259	1348	1450	905	817
1-5	1323	1415	1469	1006	920
3-4	1550	1591	1675	1283	1242
4-5	1737	1796	1838	1544	1552
1-10	1710	1736	1824	1504	1448
1-15	1959	1976	2001	1721	1775
5-10	2073	2101	2157	1979	2021
10-15	2380	2379	2357	2333	2398

Entrada % error	RC1 %	RC2 %	RC3 %	RC4 %	RC5 %
2.9-5.7	33.1	37.3	37.7	12.5	10.6
2.9-8.6	40.1	44.6	44	22.7	20.6
2.9-11.4	45.5	50	50	29.4	27.4
5.7-8.6	47.9	51.3	55.2	34.5	31.1
2.9-14.3	50.4	53.9	55.9	38.3	35
8.6-11.4	59	60.6	63.8	48.9	47.3
11.4-14.3	66.1	68.4	70	58.8	59.1
2.9-28.6	65.1	66.1	69.5	57.3	55.1
2.9-42.9	74.6	75.2	76.2	65.5	67.6
14.3-28.6	78.9	80	82.1	75.4	77
28.5-42.9	90.6	90.6	89.8	88.8	91.3

Respuestas
Año 14
No. 1
Junio 2009
ISSN 0122-820X

comportamiento que las redes RC2 y RC3 en todas las simulaciones, excepto en la última, lo cual hace pensar que cuando se utiliza más de un patrón de entrenamiento por cada carácter (situación presentada en las redes RC2 y RC3), existe una relación directa entre el número de patrones de entrenamiento y la similitud entre estos, para conseguir buena generalización.

La deficiente generalización de las redes RC2 y RC3 se debe a que estas utilizan patrones de entrenamiento aleatorios, con errores muy altos en cada uno de ellos, haciendo que en el aprendizaje no se marque la tendencia de la red que debe producir la salida correcta a valores cercanos al patrón de entrenamiento. Incluso en la red RC3, debido a que solo se introducen errores colocando bits en alto, esta no puede responder adecuadamente a entradas que contienen errores con bits en bajo, siendo por esta razón la RNA con la peor respuesta.

La red RC5 tiene para vectores con ruido de entrada bajo, un desempeño bastante notorio en comparación con las redes RC1, RC2 y RC3, y un comportamiento muy parecido para todos los niveles de ruido con la red RC4. A pesar de esto, la red RC5 mostró un menor desempeño que la red RC4 en las últimas tres simulaciones y el peor resultado de las 5 redes en la última simulación, marcando una tendencia en el error de salida de crecer a medida que se incrementa el error de la entrada; esto es debido a que, para estas entradas con errores elevados, esta red clasifica muchos de estos valores dentro de los 2 nuevos caracteres creados para el entrenamiento de esta red. La influencia de estos caracteres en el mejoramiento de la capacidad de generalización para altos niveles de error, se debe comprobar en la etapa de verificación ya que estos, no son tenidos en cuenta por la función creada en Matlab para realizar la simulación.

El error mostrado en la tabla 2, resulta ser relativo por cuanto en la verificación en hardware, la respuesta de la red RC5 es la mejor de todas las redes, dejando ver que a mayor número de patrones de entrenamiento se logra una mayor capacidad de generalización, siempre y cuando exista cierta similitud entre estos, y los valores de salida esperados.

Se debe tener en cuenta que aunque la última red entrenada tiene un buen comportamiento en cuanto a la capacidad de generalización se refiere, esta se puede mejorar realizando un entrenamiento para un juego de vectores de entrada y salida más grande, ya que en este caso solo se utilizaron 2.626 patrones de entrenamiento, equivalentes al 76,43x10exp-9 %, un valor insignificante en comparación con las 34.359'738.368 posibilidades de entrada al sistema.

Recursos utilizados en la implementación. La cantidad de recursos del FPGA utilizados en la implementación (ver tabla 3), se obtiene compilando en Project Navigator de Xilinx, el proyecto ISE que crea XSG durante la compilación bitstream. La cantidad de recursos totales del diseño se expresa de una forma general con el número equivalente en compuertas. Este parámetro es muy importante a la hora de comparar entre varios diseños que realizan la misma función pero que su descripción y por tanto su implementación se hace de manera diferente.

Tabla 3. Cantidad de recursos utilizados

DESCRIPCIÓN	UTILIZADO	DISPONIBLE	% UTILIZADO
Número de Slice Flip Flops	3.825	15.360	24
LUTs de 4 entradas	13.072	15.360	85
Número de bloques RAMs	1	24	4
Número de GCLKs	1	8	12
Equivalente en compuertas	313.578	1'000.000	31.3

Respuestas
Año 14
No.1
Junio 2009
ISSN 0122-820X

Con base en la tabla 3, se puede determinar la densidad de neuronas de la implementación. Para este caso, la densidad es de 20.905 compuertas por neurona. En este punto, se puede decir que, en el mejor de los casos, es posible realizar implementaciones en el FPGA XC3S1000 de Xilinx, con un número de neuronas cercano a 45, diferente de lo que se pensó, cuando se empezó a desarrollar este proyecto. Se deja para futuros proyectos construir una RNA con un número mayor de neuronas.

Conclusiones

La herramienta Matlab/Simulink junto con Xilinx System Generator ofrecen un ambiente de diseño amigable y conveniente para desarrollar RNAs, ya que la red es diseñada por medio de bloques e interconexiones entre estos, con un alto nivel de abstracción, puede ser simulada en software e implementada sobre FPGAs, los cuales gozan de características tales como: robustez, velocidad y paralelismo, que para las RNAs son esenciales.

Aunque no existe un método o criterio establecido que permita a los diseñadores de Redes Neuronales Artificiales seleccionar entre uno u otro tipo de red, ya que este es definido por el diseñador con base en su experiencia, la facilidad en la creación y el entrenamiento de las mismas mediante NNTool de matlab, permite, si no se tiene mucha experiencia en este tipo de diseños, elegir mediante ensayo y error la red que mejor se ajuste a la solución del problema.

Uno de los factores que permiten que una red tenga mayor capacidad de generalización, es utilizar el menor tamaño posible de la red, que proporcione un adecuado comportamiento (clasifique correctamente todos los patrones de entrenamiento), ya que entre más grande, la función que esta puede crear en forma de información entre sus neuronas, se vuelve más compleja, la cual solo es capaz de reconocer los patrones de entrenamiento y no da respuestas coherentes cuando se le presenta una entrada que no había visto antes, resultando en una deficiente generalización.

Para obtener un nivel de generalización mayor, todas las salidas esperadas de una red neuronal deben quedar definidas utilizando el menor número de bits para su codificación, a fin de evitar que la red converja a valores no deseados, que se pueden presentar en las salidas binarias de la red, que no están definidas como una salida deseada.

Para el entrenamiento de una red neuronal se debe seleccionar un número de patrones de entrada-salida coherente con el total de posibles entradas que se pueden presentar y que a su vez represente el comportamiento que se desea de la red.

Bibliografía

[1] CHAPMAN, Rob. SUTANKAYO, Steven. Implementing Artificial Neural Network Designs. Abril, 1998.

[2] LAU, Tsz Hei. Implementation of Artificial Neural Network on FPGA Devices. Nueva Zelanda, 2005. Universidad de Auckland, Departamento de ingeniería de sistemas de computo.

[3] RESTREPO, H. F. HO_MAN, R. PEREZ-URIBE, A. TEUSCHER, C. y SANCHEZ E. A networked fpga-based hardware implementation of a neural network application. En: Proceedings of the IEEE Symposium on Field Programmable Custom Computing Machines (FCCM'00), páginas 337-338, Napa, California, USA, Abril 17-19 del 2000.

[4] ACOSTA, Maria Isabel. ZULUAGA, Camilo Alfonso. Tutorial sobre redes neuronales aplicadas en Ingeniería Eléctrica y su implementación en un sitio Web. Pereira, 2000. p. 342. Universidad Tecnológica de Pereira.

Respuestas
Año 14
No. 1
Junio 2009
ISSN 0122-820X

Disponible en Internet: http://ohm.utp.edu.co/neuronales.

[5] MOCTEZUMA EUGENIO, Juan Carlos. TORRES HUITZIL, Cesar. Estudio sobre la implementación de redes neuronales artificiales usando Xilinx System Generator. Puebla, México. 2005. P.1. Benemérita Universidad Autónoma de Puebla.

[6] MOUSSA, M. AREIBI, S. y NICHOLS K. Arithmetic precision for bp networks. En: Amos Omondi and Jagath Rajapakse, editors, FPGA Implementations of Neural Networks. Kluwer Academic Publishers, Diciembre de 2003.

[7] MOORE, G.E. Cramming more components onto integrated circuits. En: Readings in computer architecture. 2000. p. 56–59. Disponible en Internet: http://portal.acm.org/citation.cfm?id=333074

[8] TOMMISCA, M.T. Efficient digital implementation of the sigmoid function for reprogrammable logic. En: Computers and Digital Techniques. Finlandia. Vol 150, no. 6 (Nov. 2003); p.403-411

[9] VIVAS, Rosa Liliana. Estudio de métodos de inteligencia artificial y desarrollo de códigos y aplicativos en Matlab para el modelamiento de sistemas basados en árboles de decisión, lógica difusa y redes neuronales. Cúcuta, 2005. P.193. Trabajo de grado en la modalidad pasantía. Universidad Francisco de Paula Santander. Plan de estudios de Ingeniería Electrónica.

[10] HAMILTON, Alberto Francisco. Estrategias de control óptimo basadas en programación dinámica y redes neuronales para sistemas MIMO continuos no lineales. España, 2002. Tesis doctoral. Universidad de la Laguna.

[11] RIVERA, Cesar Alexander. SANTIAGO, Jhon Jairo. Metodología de diseño para la implementación de Redes Neuronales Artificiales en Dispositivos Lógicos Programables. Cúcuta, 2007. p. 290. Trabajo de grado en la modalidad de proyecto de investigación. Universidad Francisco de Paula Santander. Plan de estudios de Ingeniería Electrónica.

[12] SMACH, F. ATRI, M. MITÉRAN, J. ABID, M. Design of a Neural Networks Classifier for Face Detection. En: Journal of Computer Science 2. Túnez, Francia, 2006. p. 257-260.

[13] MOLLER, M.F. A Scaled Conjugate Gradient algorithm for fast supervised learning, Neural Networks. Vol. 6, 1993. p. 525-533.