ESTUDIO SOBRE LA IMPLEMENTACIÓN DE REDES NEURONALES ARTIFICIALES USANDO XILINX SYSTEM GENERATOR

[(1)] *Juan Carlos Moctezuma Eugenio*, [(2)] *César Torres Huitzil*

[(1)] Facultad de Ciencias de la Computación, Benemérita Universidad Autónoma de Puebla, México
[(2)] Instituto Nacional de Astrofísica, Óptica y Electrónica,
Departamento de Ciencias Computacionales, Puebla-México
pumaju@ece.buap.mx

ABSTRACT

En este trabajo se realiza un estudio sobre la implementación de redes neuronales artificiales tipo feedforward y en específico de tipo perceptrón en un FPGA usando Xilinx System Generator (XSG).

El objetivo principal es ofrecer al usuario una alternativa mediante una interfaz gráfica de usuario que combina MATLAB, Simulink y XSG para desarrollar redes neuronales tipo feedforward, así como estudiar los aspectos que se deben de tomar en cuenta para la implementación hardware. Dentro de estos aspectos está el de analizar los compromisos involucrados en la representación de los números en punto fijo, como precisión, resolución, rango, etc. También se muestra una forma de calcular los recursos hardware usados para la implementación de la red.

Por otro lado se aborda de manera breve, qué es, cómo funciona y para qué sirve la herramienta Xilinx System Generator, además de las razones por la que se elige como plataforma de desarrollo.

1. INTRODUCCIÓN

El tema de las redes neuronales artificiales es un campo bastante amplio e interesante además de tener muchas aplicaciones como reconocimiento de patrones, procesamiento de señales, optimización, predicción, procesamiento de imágenes, etc.

Una Red Neuronal Artificial (RNA) es un modelo de procesamiento de información que es inspirado en el funcionamiento del sistema nervioso biológico, y de como el cerebro procesa la información. Este se compone de un gran número de elementos interconectados (neuronas) trabajando en armonía para resolver problemas específicos [1] [2]. El cerebro humano posee varias características que merecen ser imitadas por cualquier sistema electrónico, tales como [5]:

- Es robusto y tolerante a fallas.
- Es flexible y tiene la capacidad de aprendizaje.
- Puede manejar información difusa.
- Es paralelo, pequeño y compacto.

Debido a que los sistemas electrónicos deben de ser implementados a nivel hardware, y éstos serían mejores si llevaran las características del cerebro humano, es por esto que se decide hacer una aportación al tema de implementación hardware de redes neuronales. Por otro lado, 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.

La red tipo perceptrón tiene aplicaciones principalmente en la clasificación y reconocimiento de patrones, y es quizá una de las redes más sencillas pero de las más utilizadas, el perceptrón tiene la características de tener un aprendizaje supervisado, tener funciones de transferencia tipo hardlim o hardlims y de tener una topología feedforward de una o más capas [5] [3]. La etapa de aprendizaje de la red es un punto que no se aborda en este trabajo, esto es, se implementa de forma off-line y el cálculo de los pesos se obtiene mediante herramientas software como el Neural Network Toolbox de MATLAB.

El presente trabajo está dividido de la siguiente forma: en la sección 2 se hace una breve introducción a la herramienta Xilinx System Generator, la sección 3 se divide en dos partes, en la primera se realiza la red usando subsistemas y máscaras, mientras que en la segunda se hace uso de la misma metodología pero ahora usando una interfaz gráfica de usuario que permite crear de forma genérica distintas configuraciones de redes; en la sección 4 se presentan los resultados obtenidos de una aplicación a nivel hardware, la evaluación de compromisos y la estimación de recursos hardware; finalmente en la sección 5 se presentan las conclusiones.

2. XILINX SYSTEM GENERATOR

Xilinx System Generator (XSG) es un ambiente de diseño integrado (IDE) a nivel sistema para FPGAs, que utiliza Simulink, como entorno de desarrollo y se hace presente en forma de blockset. Tiene un flujo de diseño integrado, para pasar directo al archivo de configuración (*.bit) necesario para la programación del FPGA.

Una de las características más importantes de Xilinx System Generator es que posee abstracción aritmética, es decir, trabaja con representaciones en punto fijo con una precisión arbitraria, incluyendo la cuantización y el sobreflujo. También puede realizar simulaciones tanto en doble precisión como en punto fijo.

XSG puede generar automáticamente el código VHDL y un proyecto ISE del modelo que se esté desarrollando. Puede realizar síntesis jerárquica de VHDL, expansión y mapeo de hardware, además de generar archivos UCF, archivos de simulación, vectores de prueba y archivos testbench, entre otras cosas.

El Xilinx System Generator fue creado principalmente para trabajar con aplicaciones DSPs (Procesamiento Digital de Señales) complejas, pero también tiene otras aplicaciones como lo es el tema de este trabajo.

Los bloques en Xilinx System Generator operan con valores booleanos o valores arbitrarios en punto fijo. Esto es para dar una mejor aproximación a la implementación hardware. En contraste Simulink trabaja con números de punto flotante de doble precisión. La conexión entre los bloques de Xilinx System Generator y los bloques de Simulink son los bloques "gateway".

En la figura 1 se muestra a grandes rasgos el flujo de diseño de Xilinx System Generator. Como ya se había mencionado, se puede pasar directamente al archivo de configuración para programar el FPGA, por lo que la síntesis y la implementación son pasos opcionales para el usuario pero no para el System Generator [9].

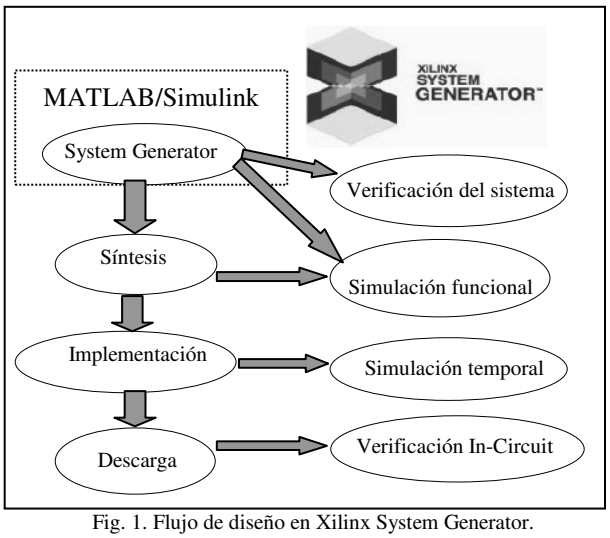

Fig. 1. Flujo de diseño en Xilinx System Generator.

3. DISEÑO

La metodología a seguir para el diseño de las redes neuronales es la siguiente: primero se realiza una red neuronal paso a paso haciendo uso de subsistemas y máscaras de Simulink, luego se realiza una interfaz gráfica configurable por el usuario en donde por medio de código de MATLAB y algunas funciones para el diseño de modelos en Simulink la red neuronal se realiza automáticamente a partir de las especificaciones.

3.1. Diseño con subsistemas y máscaras

Un subsistema es el agrupamiento de varios bloques con sus correspondientes conexiones en uno solo, formando así una "caja negra", mientras que enmascarar un subsistema se refiere a asignarle a esta caja negra un cuadro de diálogo personalizado que permita configurar diferentes parámetros de los bloques que están dentro del subsistema.

Básicamente una RNA puede ser diseñada en tres pasos: diseño de la neurona, diseño de la capa y diseño de la red multicapa. A continuación se describen estos tres procesos.

3.1.1. Diseño de la neurona.

El funcionamiento del modelo computacional de una neurona artificial está basado en tres elementos principales: los enlaces sinápticos, los cuales realizan la multiplicación de las entradas por sus pesos; un mezclador lineal, que realiza la suma ponderada de las entradas; y una función de transferencia la cual proporciona la salida de la neurona. En la figura 2 se observa la arquitectura propuesta para la neurona.

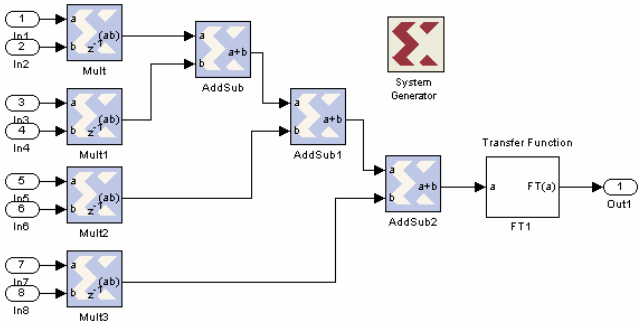

Fig. 2. Modelo de una neurona con cuatro entradas.

Como se puede observar el modelo de la neurona está conformado por multiplicadores en paralelo (que hacen los enlaces sinápticos), sumadores en cascada (que realizan la suma ponderada) y un subsistema FT (que es la función de transferencia). El bloque FT puede ser configurado para implementar una de tres posibles funciones de transferencia: 1-hardlim, 2-hardlims y 3-purelin. En la figura 3 se observa el contenido del subsistema FT, el cual tiene un bloque constante para la ganancia o *bias* y un bloque MCode que permite programar en un archivo M el

comportamiento de las diferentes funciones de transferencia. Este bloque es ideal para la implementación de funciones de transferencia ya que es muy sencillo programar el comportamiento de una función en lenguaje de alto nivel, además este lenguaje es compilado automáticamente en su correspondiente representación VHDL para poder ser sintetizado en hardware.

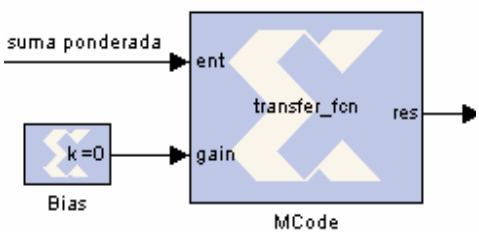

Fig. 3. Contenido del bloque FT.

Una vez que se tiene la arquitectura de la neurona, lo siguiente es realizar un subsistema de ésta, con la finalidad de tener a la neurona en un solo bloque funcional y transparente para el usuario, también se enmascara a este subsistema para configurar parámetros, como el número de bits para los sumadores o para los multiplicadores, elegir el tipo de función de transferencia, entre otros; esto para evitar la necesidad de entrar al subsistema y cambiar uno por uno estos parámetros en cada bloque. En la figura 4 se muestra el subsistema neurona.

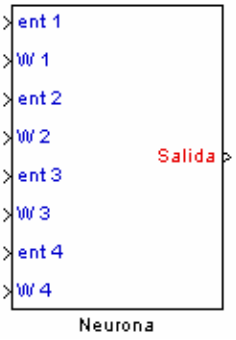

Fig. 4. Bloque funcional de la neurona de cuatro entradas.

3.1.2. Diseño de la capa.

El siguiente paso es diseñar la capa haciendo uso del bloque "neurona", realmente esto se realiza simplemente copiando la neurona tantas veces como neuronas se requieran en la capa; luego se realizan las conexiones correspondientes. En la figura 5 se muestra un ejemplo de una capa con cuatro neuronas. Una vez que se tiene el modelo de la capa, lo que hacemos después es crear nuevamente un subsistema pero ahora que agrupe a todas las neuronas y de esta forma crear un bloque "capa" funcional, como el que se observa en la figura 6.

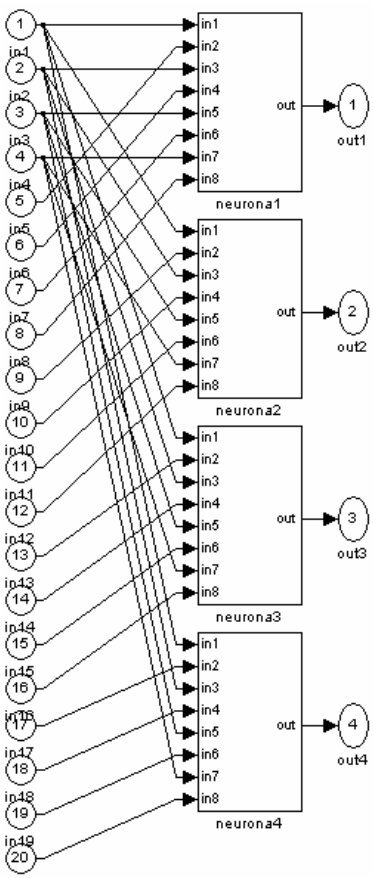

Fig. 5. Modelo de una capa.

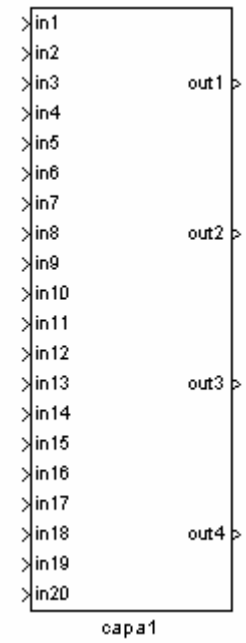

Fig. 6. Bloque funcional "capa".

3.1.3. Diseño de la red multicapa.

El último paso es diseñar la red multicapa, creando cada capa de la misma forma que en la sección 3.1.2, de esta manera el usuario puede decidir el número de neuronas y el tipo de función de transferencia en cada capa. En la figura 7 se muestra una red de 2 capas con 4 y 2 neuronas en la primera y segunda capa respectivamente, además se añaden bloques constantes para dar valor a los pesos y bloques gateways para poder simular las entradas, los gateways también sirven para implementar de forma física los puertos de E/S en los pines del FPGA.

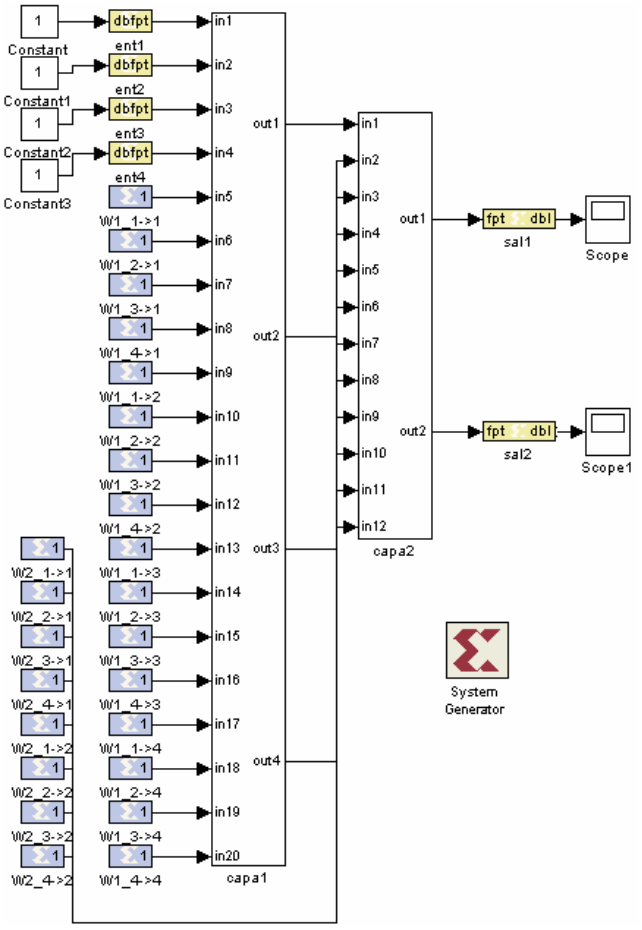

Fig. 7. Modelo de la red multicapa.

Al modelo de la figura 7 se le pueden agregar bloques constantes y bloques visualizadores de Simulink para la simulación. Por otro lado, se puede realizar un paso opcional que sería el de hacer un subsistema que encierre a todas las capas y así tener a toda la red neuronal en un solo bloque funcional.

Otro aspecto importante es que siempre que se requiera que un diseño sea sintetizado, éste debe de llevar por lo menos un bloque "System Generator", el cual permite seleccionar el dispositivo FPGA a usar, el ciclo de reloj, el archivo de configuración, etc.

3.2. Diseño usando una interfaz gráfica de usuario.

La metodología a seguir es realizar una interfaz gráfica de usuario (GUI), en la cual se configuren ciertas características de la red como: número de capas, número de neuronas, funciones de transferencia, formatos numéricos, etc. Y de esta forma, a partir de esta descripción de alto nivel, automáticamente se genere la red, hasta cierto punto se puede decir que la GUI funciona como una aplicación genérica que realiza redes neuronales. En la figura 8 se muestra la GUI propuesta.

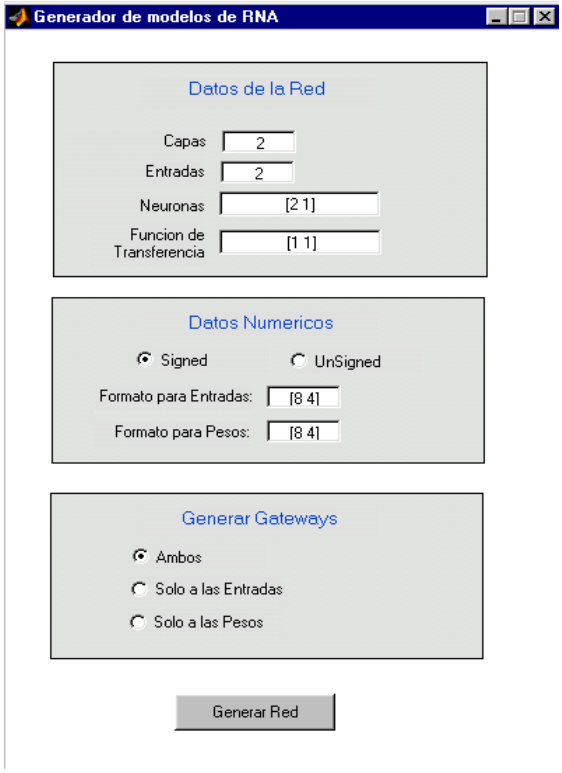

Fig. 8. GUI propuesta para la aplicación genérica de RNA.

Esta interfaz tiene tres grupos de configuración:

- Datos de la Red: en donde se indica cual va a ser la topología para la red, es decir, el número de capas, entradas, número de neuronas y funciones de transferencia en cada capa.
- Datos numéricos: aquí se indica con que tipo de números trabaja la red (signados o no signados), y el formato tanto para las entradas como para los pesos. El formato lo conforman dos números, el primero indica el número de bits y el segundo indica la posición del punto binario. Por ejemplo un formato Fix_8_6, indica un formato signado de 8 bits, de los cuales 6 bits son para la parte decimal.
- Generar gateways: genera los bloques *gateways* para la implementación hardware. Muchas veces se requiere que se implementen solo las entradas o los pesos, esta opción permite elegir entre uno y otro.

Una vez que el usuario configura todos estos campos, basta con apretar el botón de “Generar Red” para que la RNA se implemente en forma automática.

El botón *generar red* tiene asociado un *callback*, el cual se ejecuta cuando se da clic en el botón, dentro de este *callback* se escribe todo el código que hace que la red sea diseñada. En el código se usan funciones que ayudan a construir modelos en Simulink, comandos para crear bloques, configurarlos y conectarlos; también se utilizan funciones para crear subsistemas, máscaras y edición de máscaras. La interfaz de la GUI se realizó con la ayuda del GUIDE (Entorno para el Desarrollo de Interfaces Gráficas de Usuario) de MATLAB.

4. RESULTADOS

4.1. Aplicación hardware.

Para validar el modelo, se realiza una aplicación hardware que consiste en el reconocimiento de dígitos a través de un display usando una tarjeta FPGA Spartan 3. En la figura 9 se observa el diagrama a bloques de esta aplicación

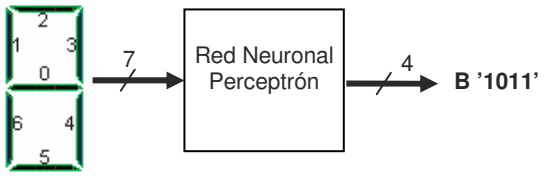

Fig. 9. RNA para el reconocimiento de dígitos mediante un display.

Como se puede ver en la figura 9, la RNA es de tipo perceptrón y tiene 7 entradas, que corresponden a cada uno de los segmentos del display; dependiendo de la representación numérica que se de en el display, la salida de la red será capaz de mostrar a dicho número pero en su equivalente binario.

El primer paso es configurar los parámetros de la GUI de la figura 8 de la siguiente forma: red de una sola capa, siete entradas, cuatro neuronas (que son las cuatro salidas), función tipo hardlim, formato Fix_5_0 para los pesos y biases, formato UFix_1_0 para las entradas, y generar gateways solamente para las entradas y salidas. En la figura 10 se muestra la red arrojada por la GUI más otros bloques que se añaden para poder realizar la implementación hardware en el FPGA.

Este modelo de implementación consta de cuatro bloques principales: el *neural network*, que es el modelo de la red perceptrón diseñada automáticamente por la GUI, el *decoder block*, que es un decodificador a siete segmentos, el *frecuency divisor*, el cual proporciona un reloj de salida con una frecuencia de 500 Hz, esta señal es usada para indicar la velocidad de barrido de las entradas de habilitación de los displays y finalmente el bloque *display controller*, que genera el barrido a las entradas de habilitación de los displays y además proporciona las salidas que van conectadas a los siete segmentos del display.

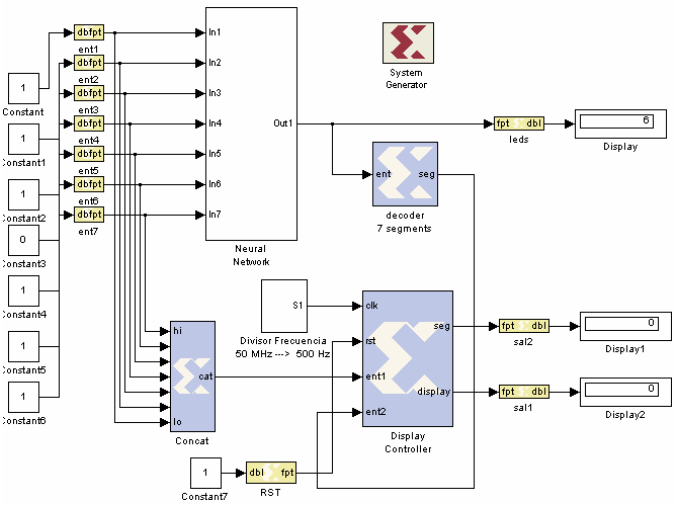

Fig. 10. Modelo de implementación para validar la RNA.

En la figura 11 se muestran los resultados de la implementación hardware para los dígitos tres, cinco y seis; para los dígitos restantes se obtienen resultados similares. El display izquierdo representa el dígito de entrada y el derecho el dígito de salida. Los leds representan la salida binaria de las neuronas.

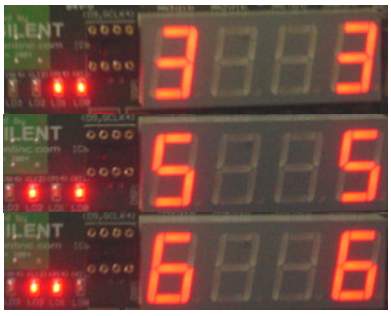

Fig. 11. Resultados de la implementación hardware.

El punto interesante, es que cuando se muestran dígitos de entrada imperfectos o con ruido, la red neuronal aproxima la salida al dígito más parecido, en la figura 12 son presentados algunos de estos casos.

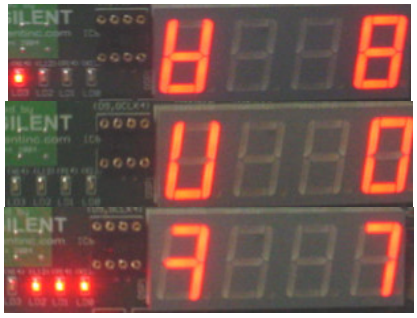

Figure 12. Resultados de la implementación hardware para dígitos de entrada con ruido.

4.2. Evaluación de compromisos.

La representación de números en una RNA es uno de los aspectos fundamentales para un buen desempeño de la red, ya que son los números los que dan sentido a las entradas, los pesos, los bias y las salidas de una red. Por esta razón, en esta sección se presentan algunos conceptos y ecuaciones que son de gran ayuda para elegir un buen formato numérico con el que trabajará la red neuronal. El formato que usa System Generator es en punto fijo y la notación es la siguiente:

Fix_wd_pt ⟹ Reales Signed (wd ≥ pt)

UFix_wd_pt ⟹ Reales Unsigned (wd ≥ pt)

wd = ancho de palabra (número de bits)
pt = posición del punto binario hacia la izquierda del LSB

4.2.1. Precisión.

La precisión esta definida como el número de bits que se usan para representar los números.

$$\text{Precisión} = wd \tag{1}$$

4.2.2. Resolución.

La resolución significa el número más pequeño (que no sea cero) que se puede representar. En otras palabras, es la porción en la que se pueden ir incrementando los números con un formato dado.

$$\text{Resolución} = 1 / 2^{pt} \tag{2}$$

4.2.3. Rango.

El rango proporciona el intervalo de números que pueden ser representados. Se puede hacer uso del rango y de la resolución para obtener toda la gama de números que se pueden representar con un formato numérico.

$$\text{Rango\_UFix} = [\,0 \quad (2^{wd} - 1) / 2^{pt}\,] \tag{3}$$

$$\text{Rango\_Fix} = [\,-(2^{wd-1}) \quad (2^{wd-1} - 1)\,] / 2^{pt} \tag{4}$$

4.2.4. Rango dinámico.

El rango dinámico (RD) es la razón entre el máximo valor absoluto representable y el mínimo valor absoluto (que no sea cero) representable. Éste da una idea de la capacidad que tiene la red para representar distintas cantidades.

$$\text{RD\_UFix} = 2^{wd} - 1 \tag{5}$$

$$\text{RD\_Fix} = 2^{wd-1} \tag{6}$$

4.2.5. Sobreflujo y Cuantización.

Cuando se trabaja con formatos limitados en bits suelen ocurrir errores de sobreflujo y cuantización. El sobreflujo es el efecto producido cuando se quiere representar una cantidad en la parte entera que esta fuera del rango de valores admitidos. Mientras que cuando el rango de valores en punto fijo no satisface la representación para la parte fraccionaria se dice que hay cuantización. En la figura 13 se visualizan estos efectos.

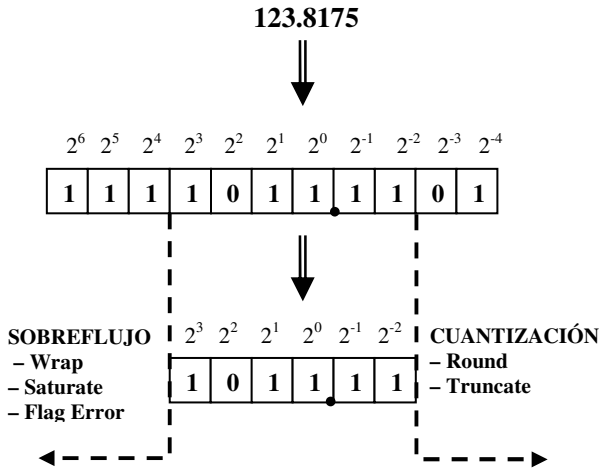

Figure 13. Efectos del sobreflujo y la cuantización en una representación en punto fijo.

4.3. Estimación de recursos hardware.

En esta sección se muestra como calcular de una forma rápida la estimación de recursos hardware para una RNA específica o para un diseño en general.

El bloque "Resource Estimator" proporciona una estimación rápida de recursos FPGA requerida para la implementación de un modelo o subsistema hecho en XSG. Esta estimación permite ver como se afectan los requerimientos hardware cuando se cambian ciertos parámetros en el diseño sin necesidad de realizar nuevamente todo el proceso que involucra a dicha estimación; como la síntesis, mapeo y ruteo.

Las estimaciones son hechas cuando se invocan los "estimadores de bloques específicos", los cuales calculan los recursos hardware que se necesitan para implementar a cada uno de los bloques de Xilinx, luego estas

estimaciones individuales se suman para obtener un agregado de la estimación total de recursos. La estimación de estos recursos se realiza para siete campos: Slices (generalmente compuestos por 2 flipflops, 2 LUTs, algunos multiplexores y una pequeña lógica de control), Lookup Tables (LUTs), FlipFlops (FFs), Bloques de memoria (BRAM), multiplicadores 18x18, Buffers Tri-estado y puertos de entrada/salida (IOBs). En la figura 14 se observa la ventana para el bloque Resource Estimator.

Por ejemplo, para la aplicación de la sección 4.1, se obtuvieron los siguientes recursos: 454 Slices, 448 FFs, 621 LUTs y 23 IOBs, que en promedio representan aproximadamente un 15% de los recursos totales del FPGA. Cabe mencionar que estos recursos no solo son de la red neuronal, sino también del decodificador, del divisor de frecuencia y del controlador del display.

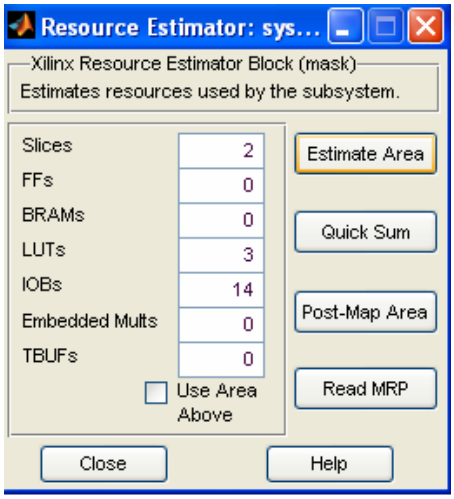

Figure 14. Ventana para el bloque Resource Estimator

5. CONCLUSIONES

La implementación de redes neuronales en lógica programable ofrece importantes características y una buena alternativa para el diseño y desarrollo de las mismas, las plataformas tanto software como hardware que proporciona la lógica programable son ideales para tener una mejor aproximación a las redes neuronales biológicas, lo cual es fundamental en los modelos artificiales.

La herramienta Xilinx System Generator, es novedosa en aplicaciones de redes neuronales, además ofrece un ambiente de diseño amigable y conveniente para desarrollar RNAs, ya que la red es diseñada por medio de bloques e interconexiones entre ellos, también tiene la capacidad de trabajar con números reales con representación en punto fijo y además posee un alto nivel de abstracción en sus bloques. Por otro lado, esta herramienta soporta la simulación software, pero lo más importante, es que se pueden realizar implementaciones hardware sobre FPGAs, los cuales tienen características tales como: robustez, velocidad y paralelismo; que en RNA son esenciales.

La realización de "aplicaciones genéricas" para redes neuronales facilita el prototipo y la simulación de diferentes modelos antes de ser implementados en hardware, un ejemplo es la GUI que aquí se presenta.

La elección de la aritmética empleada y la precisión, proporcionan información del rango de números representable, el rango dinámico, la resolución y la exactitud en el funcionamiento de la red. Por ejemplo, la elección de una precisión alta se verá reflejada en un mejor desempeño de la red, en cuanto a que se pueden representar una mayor cantidad de números y con una buena resolución, pero también refleja un incremento de recursos hardware.

Por lo tanto se debe de encontrar un punto de equilibrio entre la precisión de los resultados y el desempeño de la red. En este trabajo, se comprobó que un estudio teórico previo de las características de la red neuronal enfocado a la representación de números es una buena opción para encontrar dicho equilibrio.

6. REFERENCIAS

[1] Patrick van der Smagt, Ben Krose, "An Introduction to Neuronal Networks", octava edición. 1996.

[2] Martín del Brío Bonifacio, Sanz Molina Alfredo , "Redes Neuronales y Sistemas Difusos", Editorial Alfaomega, 2002.

[3] Haykin Simon, "Neural Networks. A Comprehensive Foundation", Edit. Prentice Hall, Primera Edición, 1994.

[4] Erick L. Oberstar, "Fixed Point Representation and Fractional Math", Julio 2004.

[5] Tutorial de Redes Neuronales, http://ohm.utp.edu.co/neuronales

[6] Randy Yates, "Fixed-Point Arithmetic: An Introduction", March 3, 2001.

[7] K.C. Chang, "Digital Design and Modeling with VHDL and Synthesis", IEEE Computer Society Press, 1997.

[8] Liao Ylhua , "Neural Networks in Hardware: A Survey", Department of Computer Science, University of California.

[9] Xilinx System Generator User's Guide, www.xilinx.com

[10] "Spartan-3 Starter Kit Board Guide", www.digileninc.com

[11] "FPGA Spartan-3 Datasheet", www.xilinx.com

[12] MATLAB website, www.mathworks.com