IMPLEMENTACIÓN HARDWARE DE FUNCIONES DE TRANSFERENCIA PARA REDES NEURONALES ARTIFICIALES

J.C. Moctezuma Eugenio, A. Sánchez Galvez, A. Ata Pérez

Facultad de Ciencias de la Computación, Benemérita Universidad Autónoma de Puebla, México

moctezumajc@yahoo.com

ABSTRACT

En este trabajo se presenta la implementación hardware en FPGA de funciones de transferencia para redes neuronales artificiales usando lógica programable y en específico la herramienta Xilinx System Generator. Se usa código de MATLAB sintetizable para realizar funciones de tipo umbral y "piecewise linear", mientras que para las funciones no lineales se hace uso de memorias ROM distribuidas o dedicadas.

Como plataforma de diseño para estas funciones se realiza un bloque tipo Simulink, el cual permite implementar uno de diez principales tipos de funciones de transferencia. Finalmente se realiza un estudio de los recursos hardware y de la precisión de los resultados en el FPGA comparados con los proporcionados por MATLAB.

1. INTRODUCTION

Las Redes Neuronales Artificiales (RNA) son sistemas de computación inspirados en el funcionamiento del cerebro humano, están compuestos por una gran cantidad de elementos simples de procesamiento (neuronas) conectados entre sí y que operan de forma masivamente paralela. En los últimos diez años han retomado un gran auge, ya que consiguen resolver problemas relacionados con el reconocimiento de patrones, predicción, codificación, clasificación, control, optimización, etc. Por lo que su implementación hardware es una parte esencial para el desarrollo de estas aplicaciones [1] [2].

Las neuronas artificiales están compuestas de tres elementos principales que son: lazos sinápticos, un mezclador lineal y una función de transferencia, ésta última es de especial interés ya que puede proporcionar características lineales y no lineales al comportamiento de una RNA [3]. Este trabajo se enfoca a la realización de estas funciones de transferencia en hardware y usando herramientas de lógica programable. La forma de diseño es por medio de un bloque en Simulink sintetizable, que puede implementar hasta diez diferentes tipos de funciones.

En la literatura se han encontrado diferentes formas de implementación, por ejemplo, con comparadores, con algún lenguaje HDL, con aproximaciones lineales, con look-up tables, etc. Aquí se hace uso de algunas de estas técnicas, pero integradas en un bloque genérico hecho con la herramienta Xilinx System Generator, y que hacen posible su implementación en FPGAs de una manera rápida y sencilla.

2. FUNCIONES DE TRANSFERENCIA

La función de transferencia o de activación se encarga de calcular el nivel de activación de la neurona en función de la entrada total, también denota la salida de la neurona. Se pueden identificar tres tipos básicos de funciones de transferencia: las funciones de umbral, en las cuales la salida es un valor discreto que supera o no un determinado umbral, existen también las funciones lineales o al menos en una parte (picewise linear), y finalmente están las funciones no lineales en donde las "funciones sigmoidales" son las más habituales.

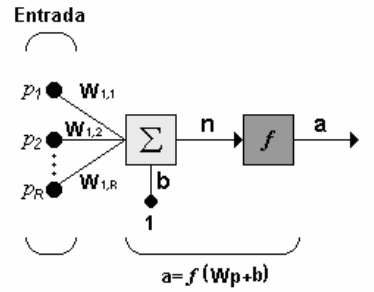

Fig. 1. Modelo computacional de una neurona. La salida total '*a*' de la neurona esta dada por el resultado de la función de transferencia aplicada a la variable 'n', la cual es la suma ponderada de sus entradas.

Básicamente, las funciones de transferencia realizan dos tareas importantes: la primera, es que sirven para limitar la salida de una neurona, y así los resultados no crezcan a valores demasiado grandes; y la segunda, es que

proporciona características de no linealidad, lo cual es muy importante en RNA. En la tabla 1, se muestran los diez tipos de funciones de transferencia más usados y que además, son los que se implementan en este trabajo.

TABLA 1.PRINCIPALES TIPOS DE FUNCIONES DE TRANSFERENCIA

Nombre	Símbolo	Relación Entrada / Salida
1. hardlim		**a = 0 n < 0** **a = 1 n ≥ 0**
2. hardlims		**a = – 1 n < 0** **a = 1 n ≥ 0**
3. poslin		**a = 0 n < 0** **a = n n ≥ 0**
4 .purelin		**a = n**
5. satlin		**a = 0 n < 0** **a = n 0 ≤ n ≤1** **a = 1 n > 1**
6. satlins		**a = – 1 n < – 1** **a = n – 1 ≤ n ≤1** **a = 1 n > 1**
7. tribas		**a = 1–|n| –1< n < 1** **a = 0 otro caso**
8. logsig		$a = \frac{1}{1+e^{-n}}$
9. tansig		$a = \frac{e^{n} - e^{-n}}{e^{n} + e^{-n}}$
10. radbas		$a = e^{-n^2}$

3. DISEÑO

La plataforma de desarrollo para este trabajo es Xilinx System Generator (XSG), el cual es un ambiente de diseño a nivel sistema (IDE) para FPGAs, se incorpora a Simulink en forma de plug-in y por lo tanto lo usa como entorno de desarrollo, además se hace presente en forma de Blockset. Tiene un flujo de diseño integrado, para pasar directo al archivo de configuración bitstream (*.bit) necesario para la programación del FPGA.

3.1. Diseño usando el bloque MCode.

El bloque MCode funciona como una caja negra, la cual contiene código M de MATLAB. El bloque ejecuta el código y calcula los valores que serán entregados a los bloques de Simulink durante la simulación. Este mismo código es traducido a una forma funcional equivalente en VHDL para que el hardware pueda ser generado. A continuación se muestra un ejemplo de código M para implementar la función *poslin*.

```
function  res = funtransfer2(ent)
        if (ent < 0)
           aux = 0;
        else
           aux = ent;
        end
    res = xfix({xlSigned, 8, 4},aux);
```

Como se puede observar es muy fácil realizar funciones que involucren comparaciones, sentencias condicionales y asignaciones simples de números enteros, como lo son las funciones de umbral y las de tipo picewise linear. En la figura 2 se observa la forma en como este bloque es presentado al usuario.

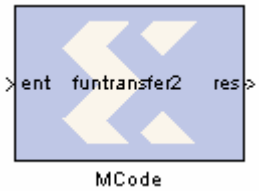

Fig. 2. Bloque MCode cuyo comportamiento esta gobernado por un archivo que contiene código M de MATLAB.

3.2. Diseño usando bloques BRAMs

Generalmente existen dos opciones para la realización de funciones no lineales, una de ellas es por medio de aproximaciones lineales, pero esto puede llevar a errores en puntos críticos de la función; la otra opción es realizar una tabla (por medio de look-up tables) de valores discretos con su correspondiente evaluación de la función no lineal, es decir, las funciones se realizan en forma de tablas, guardadas en bloques ROM de Xilinx. En general estos bloques tienen las siguientes características:

- El número de localidades de memoria depende del número de bits que se tenga a la entrada. Con esto se evita el desperdicio de localidades innecesarias, o bien, se evita la falta de ellas.
- Se usan las funciones de transferencia del Neural Network Toolbox de MATLAB para la inicialización de la ROM, con esto, no es necesario la declaración matemática explícita de la función.
- El formato para la salida es ajustable por el usuario.
- El valor de entrada funciona como apuntador hacia la localidad de memoria que contenga el valor deseado.

3.3. Realización del bloque en Simulink

La idea general es realizar un bloque tipo Simulink, el cual pueda implementar cualquiera de las diez funciones de transferencia de la tabla 1. Por medio de una máscara asociada a este bloque, el usuario podrá seleccionar y configurar distintos parámetros.

El bloque en Simulink, en realidad es un subsistema, llamado "FT", que tiene una entrada y una salida, este subsistema encierra al bloque MCode en caso de que se halla elegido las funciones del 1 al 7, o bien, al bloque ROM en caso de que se elijan del 8 al 10.

En la figura 3.a se muestra el contenido del subsistema FT para las funciones 1-7, mientras que en la figura 3.b para las funciones 8-10. Por último en la figura 4, se muestra el subsistema FT como bloque configurable por el usuario y totalmente sintetizable en hardware, por lo tanto, cualquier diseño de redes neuronales en Xilinx System Generator puede usar este bloque, además éste puede compilarse para generar archivos VHDL y EDIF, lo que le da un buen nivel de portabilidad a este diseño.

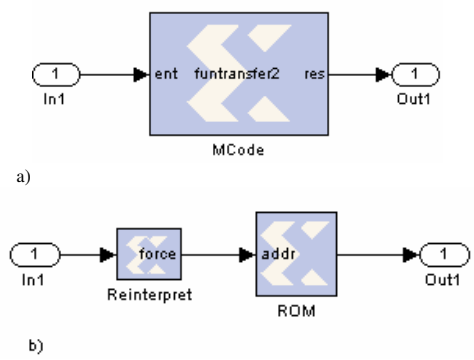

Fig. 3. Contenido del subsistema FT. a) contenido en caso de implementar funciones de tipo umbral y lineal. b) contenido en caso de implementar funciones de tipo no lineal.

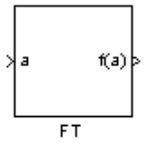

Fig. 4. Subsistema o bloque final FT.

4. RESULTADOS

En la figura 5, se observa el modelo hecho en Simulink para la validación del bloque FT

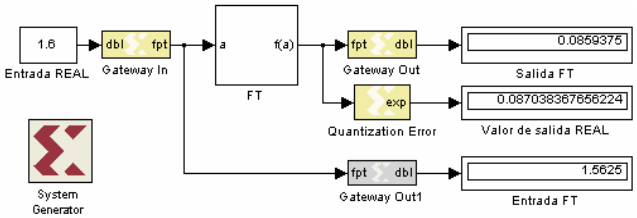

Fig. 5. Diseño en Simulink para la validación del bloque FT

Ya que los bloques de Xilinx trabajan con números en punto fijo, entonces se tiene cierta limitación para la representación de valores reales, por lo que suelen ocurrir errores de Sobreflujo y Cuantización. La elección del tipo de Sobreflujo y Cuantización depende de las necesidades y hasta cierto punto del gusto del diseñador, para este trabajo, se elige la opción de *Saturate* para el Sobreflujo, la cual satura hacia el número más grande, mientras que para la Cuantización se elige la opción *Round*, que como su nombre indica, redondea hacia el valor más próximo.

Con respecto a las funciones de umbral (1 y 2 de la tabla 1), realmente no existe mayor dificultad en la precisión de sus resultados, ya que éstos son valores enteros que no ocupan muchos bits para su representación. Para las funciones lineales o picewise linear (3, 4, 5, 6 y 7), la exactitud de los resultados dependen directamente de los errores de Sobreflujo y Cuantización de la entrada. Para las funciones no lineales (8,9 y 10), la precisión de los resultados depende de los errores de sobreflujo y cuantización tanto de la entrada como de la salida. En la tabla 2, se observan los resultados para diferentes funciones de transferencia.

TABLA 2. RESULTADOS PARA DISTINTAS FUNCIONES DE TRANSFERENCIA

Entrada REAL	Entrada al bloque FT	Salida IDEAL	Salida del bloque FT
-2	-2	-1	-1
-0.99	-0.9921875	-0.99	-0.9921875
-0.65	-0.6484375	-0.65	-0.6484375
0.158	0.15625	0.158	0.15625
0.88	0.88281525	0.88	0.8828125
10	3.9921875	1	1

a) Resultados para la función *satlins*, configuración de entrada: Signed, 10 bits, 7 bits para la parte decimal; configuración para la salida: Signed, 10 bits y 8 para la parte decimal.

-5.3	-4	-0.99995016	-1
-1.45	-1.453125	-0.89569287	-0.894531
-0.125	-0.125	-0.124353	-0.125
0.422	0.421875	0.39861398	0.3984375
1	1	0.76159415	0.76171875
2.75	2.75	0.99185972	0.9921875

b) Resultados para la función *tansig*, configuración de entrada: Signed, 10 bits, 7 bits para la parte decimal; configuración para la salida: Unsigned, 10 bits y 8 para la parte decimal.

-1.6	-1.6015625	0.07730474	0.07617187
-0.72	-0.71875	0.59547254	0.59570312
-0.05	-0.046875	0.99750312	0.99804687
0.265	0.265625	0.93218405	0.93164062
0.5	0.5	0.77880078	0.77929687
2.5	2.5	0.00193045	0.00195312

c) Resultados para la función *radbas*, configuración de entrada: Signed, 10 bits, 7 bits para la parte decimal; configuración para la salida: Unsigned, 10 bits y 9 para la parte decimal.

Por otro lado, los diseños hechos en Xilinx System Generator son Bit-true & Cycle-true, esto garantiza que los

resultados obtenidos en simulación serán los mismos que los obtenidos por la implementación hardware. Se usa una tarjeta de desarrollo FPGA Spartan-3 para la validación del diseño. En la figura 6 se muestra la implementación de la función log*sig*, para una entrada con formato fixed_8_6 de valor 1.6875 (en binario: '01.101100'), la cual da una salida con formato Ufixed_8_7 de valor 0.8438951 (en binario: '0.1101100').

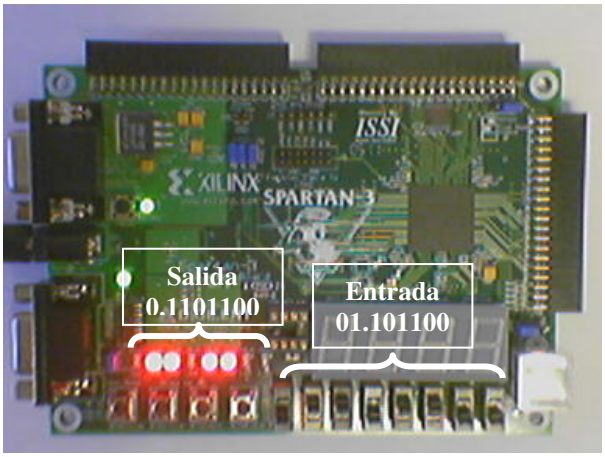

Fig. 6. Tarjeta de desarrollo FPGA Spartan-3 para la validación hardware del diseño. Los switches es la entradas y los leds la salida. Ambos están representados en punto fijo.

En la tabla 3 se muestra el consumo de recursos hardware para diferentes funciones de transferencia. En esta tabla se observa que los recursos son mínimos cuando se implementan con bloques Mcode. Por lo que respecta a las funciones implementadas con bloques RAM del FPGA, existen dos puntos a considerar: si la función es realizada con memoria distribuida (en caso de que la familia de FPGA no cuente con bloques de memoria dedicados), entonces se consumen más recursos, en cuanto a LUTs, Slices y Flip-Flops, sin embargo si se realiza con bloques RAM dedicados, entonces solamente se usa un bloque RAM.

TABLA 3. RECURSOS HARDWARE PARA DISTINTAS FUNCIONES DE TRANSFERENCIA

Función	Configuración	Slices	LUTs	IOBs
Hardlim	1	1	1	2
	2	1	1	2
Hardlims	3	1	1	3
	4	1	1	3
Poslin	5	4	7	16
	6	4	7	16
Purelin	5	0	0	16
	6	0	0	16
Satlin	5	3	6	16
	6	3	5	16
Satlins	5	3	6	16
	6	3	5	16
Tribas	5	9	16	16
	6	15	25	16
Logsig	7	154	307	20
	8	154	307	20
Tansig	7	154	307	20
	8	154	307	20
Radbas	7	154	307	20
	8	154	307	20

Configuración 1. Entrada: Fix_8_5, Salida: Boolean. Configuración 2. Entrada: Fix_8_2, Salida Boolean. Configuración 3. Entrada: Fix_8_5, Salida: Fix_2_0, Configuración 4. Entrada: Fix_8_2, Salida: Fix_2_0. Configuración 5. Entrada: Fix_8_5, Salida: Fix_8_5. Configuración 6. Entrada: Fix_8_1, Salida: Fix_8_1. Configuración 7. Entrada: Fix_10_8, Salida: Fix_10_8. Configuración 8. Entrada: Fix_10_4, Salida: Fix_10_4.

5. CONCLUSIONES

El bloque propuesto en este trabajo es una buena alternativa para implementar distintos tipos de funciones de transferencia por varias razones: es genérico, es decir, se puede generar una de las diez funciones de transferencia más usadas, es fácil de usar, tiene una interfaz gráfica, puede manejar números reales, es importable a VHDL y lo más importante es que es sintetizable en FPGAs con un bajo consumo de recursos hardware.

La precisión de los resultados depende directamente de las tres principales características del formato numérico: aritmética empleada (Signado, no Signado ó Booleano), número de bits y posición del punto binario.

El gasto de recursos hardware para funciones de tipo umbral o picewise linear es casi nulo, mientras que para las funciones no lineales, se recomienda usar bloques de memoria para una mayor precisión en los resultados

7. REFERENCIAS

[1] José R. Hilera, Victor J. Martínez, "Redes Neuronales Artificiales, fundamentos, modelos y aplicaciones" Edit. Alfaomega, 2000.

[2] Bonifacio Martín, Alfredo Sanz, "Redes Neuronales y Sistemas Difusos", Edit. Alfaomega, 2002.

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

[4] Yates Randy, "Fixed-Point Arithmetic: An Introduction", 2001.

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

[6] "Tutorial de Redes Neuronales Artificiales", Universidad Tecnológica de Pereira, http://ohm.utp.edu.co/neuronales/

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

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

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