FPGA DESIGN OF A MLP ARTIFICIAL NEURAL NETWORK ARCHITECTURE

Antonyus Ferreira, Edna Barros, Teresa Ludermir
{apaf, ensb, tbl}@cin.ufpe.br

Informatics Center, Federal University of Pernambuco, Brazil

ABSTRACT

This work presents the challenges and proposed solutions on implementing a FPGA based architecture of a Multilayer Perceptron (MLP) Artificial Neural Network (ANN). Choices like switching between to use either float point arithmetic or fixed point are mentioned and comparisons with other architectures as well. The scope of this paper does not include the ANNs learning phase.

1. INTRODUCTION

Nowadays several embedded devices include some functionality based on artificial intelligence (AI) techniques. And some embedded systems' requirements conflicts with AI features like parallel processing. For example to introduce an ANN in a digital camera with classical implementation of ANNs or the power consumption would raise either the response would take too much time to be computed. And in the other hand hardware implementations of these techniques could bring more value to this class of products. In this purpose are aligned the works of [2], [7], [8], [9] e [10].

Following those motivations we describe in chapters 2 and 3 some important concepts about ANNs and FPGA respectively. In chapter 4 are presented the implementation flow and some choices taken on the design of implementing MLP ANNs in FPGA. Results come in chapter 5. Chapter 6 we compare our implementation with two other works.

2. MLP ARTIFICIAL NEURAL NETWORKS

All the animals' brains are compounded by billions of cells interconnected in a giant net. And ANNs are computational models whose organization and architecture are inspired in animals' brains structure. This model inherits from biological model its parallel and distributed feature.

ANNs can be found in many areas like signal processing, medical image analysis, diagnostic systems and time series forecasting. Some desired properties [1] of ANNs are:

a. Learning through examples
   -Non parametric statistical inference
b. Adaptability
c. Generalization
d. Fault tolerance
e. Fast implementation

2.1 Artificial Neuron

An artificial neuron is the unit of the neural networks architecture. In the neurons' structure can be found:

a. An input set that receives neurons' input signals;
b. A synaptic set whose intensity is represented by an associated weight;
c. An activation function that compares inputs and their synaptic with the threshold function to define neuron's output.

In the following figure [2] each *Wi* represents the weights associated with each *Xi* and *Φ* is the activation function. The result of the synaptic is given by the inner product (*u*) of the inputs vector by the weight vector and the output by the computation of *Φ(u)*.

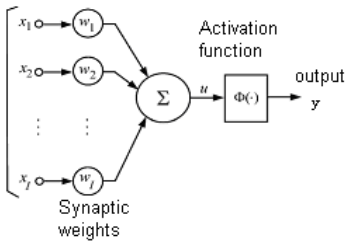

Figure 1 Artificial neuron

Some activation functions used are:

a. Step function;
   $\Phi(u) = 1$ if $u > 0$, $\Phi(u) = 0$, otherwise
b. Ramp function
   $\Phi(u) = \max\{0.0, \min\{1.0, u + 0.5\}\}$
c. Sigmoid function
   $\Phi(u) = a / \{ 1 + \exp(-bu) \}$

2.2 Back-Propagation algorithm

It is the most known training algorithm of ANNs and is based on supervised learning (pairs input – desired output to adjust the net's weights). The training has two phases called forward e backward.

In the first phase the inputs are computed through the neurons layers and its outputs are compared with the desired output. And in the backward phase in each neuron is made an adjustment to minimize the output error.

3. FPGA: A SHORT INTRODUCTION

Field Programmable Gate Arrays (FPGAs) are compounded by a matrix of logic blocks that can be connected ones to each others to implement complex logic expressions [5]. The user design is accomplished

specifying simple logic expressions to each cell and selectiveness closing connections in the block matrix. See picture above.

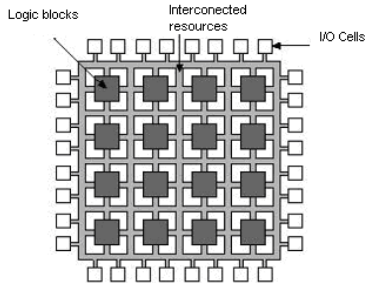

Figure 2 FPGA architecture

Nowadays, FPGAs are used for fast digital circuit prototyping. And so these systems could be produced in large scale as ASICs.

4. FPGA DESIGN OF MLP RNAS

4.1 Floating point Vs Fixed-point

The float point representation is similar to the scientific notation where the number is multiplied by its base raised by an exponent. The great benefit of representing a number in the float point pattern (IEEE 754) is to provide several precision degrees based on the used scale.

In other side, the fixed-point notation defines a specific radix and there is a fixed number of bits to represent the integer part and the fractionary part. To implement operations in this representation is as easy and fast as manipulate integer numbers. A comparison with the two approaches could be seen in the chart bellow.

Floating point	Fixed-point
Accuracy	Product's final cost
Dynamic Range	Speed
Shorter design time	

Due to add more fidelity to the original model of ANNs we decided to use float point arithmetic.

This choice allows working with number of different magnitudes and guarantees that the originals ANNs learning algorithms could be used without any problems.

Altera provides float point components in its IP library. We used one adder (ALTFP_ADD_SUB), one multiplier (ALTFP_MULT), one comparator (ALTFP_COMPARE) all of them following single precision IEEE 754 pattern and using VHDL (as well in entire project).

4.2 Activation function's approximation

Although direct implementation of the sigmoid is suitable, it is requires excessive logic use. We are intending to use a more intelligent approach than to implement a non accurate look up table.

The first alternative defines the functions:

$$y_1(x) = 0 \quad y_2(x) = \frac{1}{2}\left(1 + \frac{x}{2}\right) \quad y_3(x) = 1$$

The method iteratively computes the output within about 4 steps as described by [13]. It represents an approximation mean error of 1.4539e-017 and a max error of 0.0194 in the interval [-5,5].

A second method presents a linear by parts approximation [12]:

Tabela 1 Approximation 2

Operação	Condição
$Y = 1$	$\|X\| \geq 5$
$Y = 0.03125 \cdot \|X\| + 0.84375$	$2.375 \leq \|X\| < 5$
$Y = 0.125 \cdot \|X\| + 0.625$	$1 \leq \|X\| < 2.375$
$Y = 0.25 \cdot \|X\| + 0.5$	$0 \leq \|X\| < 1$

The disadvantage of this approach is it is not smooth. And a activation function must be differentiable to allow the application of an descendent gradient learning algorithm. Although this method seems to be inaccurate it results in a mean error of 8.9214e-018 and a max error of 0.0189

The last approach tested is classified as a quadratic defined by parts approximation. Zhang, Vassiliadis e Delgado–Frias [11] propose a method using only one multiplication.

$$y = \begin{cases} 2^{-1} * (1 - |2^{-2} * x|)^2 & -4 < x < 0 \\ 1 - 2^{-1} * (1 - |2^{-2} * x|)^2 & 0 \leq x < 4 \end{cases}$$

This approximation represents a smooth and fast method. Its mean error is 8.5910e-018 and max error 0.0215.

In the figure 3 we can see visually how each approximation behavior. The methods are in order from left to right.

Method	Mean error	max error	smooth	fast
1º order iterative	1.4539e-017	0.0194	yes	no
1º order by parts	8.9214e-018	0.0189	no	yes
2º order by parts	8.5910e-018	0.0215	yes	yes

The third method has a higher max error but it is smooth and easily implemented with little logic consumption so we decide to use it.

4.3 Neuron's implementation

We have chosen how neuron function activation will be computed. Now it is left to define the propagation rule of the neuron. Due to area considerations, only one float point multiplier and adder were used. Nevertheless some parallelism could be introduced in the calculi of the sum of products as shown on the next chart.

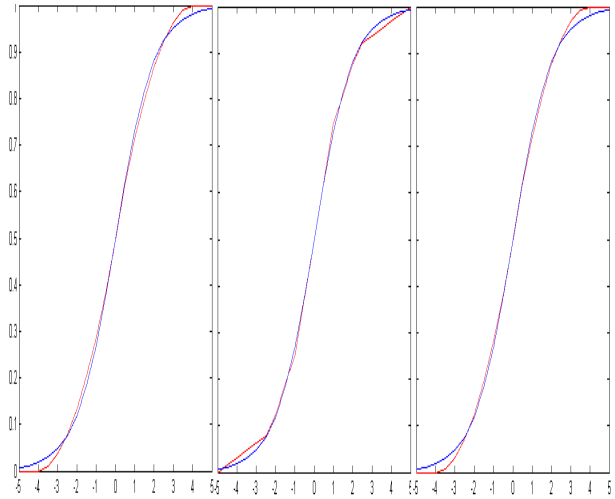

Figure 3 Sigmoid approximations (real sigmoid in blue)

Tabela 2 Parallelism on computing the net

Inputs	function	Operations
2	X1 x W1 + X2 x W2 + W0	X X + +
3	X1xW1 + X2xW2 + X3xW3 + W0	X X X + + +
4	X1xW1 + X2xW2 + X3xW3 + X4xW4 +W0	X X X X + + + +

Where Xi is an input, Wi is a weight associated to each input. The operations column vertically shows the time when each computation is done. So we can do one sum and multiplication at the same time.

The component neuron with e 2 inputs is like this:

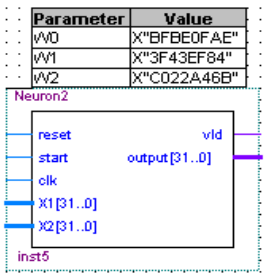

Figure 4 Two inputs Neuron

Above the component can be viewed an editable chart where the user might enter the weight of each input. Therefore the user can use the component with no knowledge about its implementation.

4.4 Sigmoid implementation

Following the steps we computed the response of each neuron:

1. Compare $|net| > 4$ (out of approximation range)
   a. If true then saturate output to 0 or 1.
   b. else step 2
2. Right shift net's exponent (multiply by 2^{-2}) and reset the signal bit
3. Compute *1 - result2* (result of step 2)
4. Then raise *result3* by *2*
5. Right shift the *result4* (multiply by 2^{-1})
6. If *net < 0* then **done**
   else do *1 – result5*

4.5 ANN's implementation

To implement a neural network with our neuron users only need to instantiate the neurons and connect them.

Additionally a control module determines the activation of each neurons layer. Two implementations of the controller were tested, one sequential and another allowing the neurons layers compute at the same time taking hands of the parallelism of the model.

5. RESULTS

We used the diabetes problem to validate our architecture. The dimension of the problem was reduced from eight to five and with up to 384 examples vectors. So the net has 5 input neurons, 2 intermediate neurons in the hidden layer and 2 output neurons (5:2:2).

First of all we had to compare the performance of the hardware implementation with the software implementation in C++. In the software execution we used an AMD Athlon 64 3200+ 2.20GHz computer with 512 MB of memory.

The ANN in FPGA (sequential implementation) computed all the 384 examples within 299,45µs versus 23ms in software i.e. 76,8 times faster. The parallelized version in hardware spent 165,38µs which is 139 times faster than software implementation.

The resource utilization were compatible with the choice of using float point. We used an altera STRATIX II FPGA (EP2S60F672C5ES) speed grade -5. The entire neural network in the study case (with the test structure) totalized 6.692 Combinational ALUTs, 5.447 Registers, 114.688 memory blocks (5%) e 32 9-Bit DSP blocks (11%).

6. RELATED WORK

6.1 FPGA implementation of a face detector using neural networks [3]

Yongsoon Lee e Seok-Bum Ko in this work used floating point arithmetic due to same causes that guided us in our choice and implemented an MLP ANN. They chose the follow approximation of the sigmoid that is clearly slower than ours.

$$\frac{1}{1- e^{-SUM}} = \frac{1}{1+1/k} = \frac{k}{k+1},$$

$$, where \;\; k = (1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \frac{x^4}{24}).$$

They obtained a fmax of 38MHz versus 160MHz of our model. But this work makes evident RNA utilization in real time applications.

6.2 FPGA Implementation of a Neural Network for a Real-Time Hand Tracking System [16]

The authors implemented the neuron with fixed point adders and multipliers. They differ using hyperbolic tangent as the activation function and its approximation using a look up table obviously targeting meet performance requirements. But the look up table use only 15 levels as figure 6 shows.

Although the use of float point neuron our mean performance was higher than that indicated by the authors: 43,07ns (our) versus 71ns.

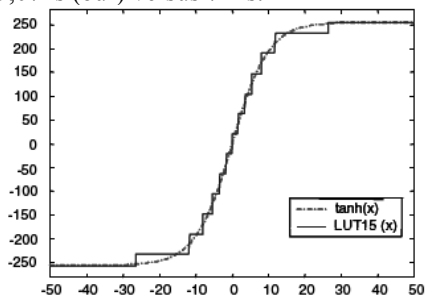

Figure 5 look up table approximation for tanh

7. CONCLUSIONS AND FUTURE WORKS

Comparatively we demonstrate that our choices were efficient on designing ANNs in FPGA. Although the learning phase of the ANNs was not in the scope of this work we were cautious on thinking of a future implementation with learning phase.

We intend to build an ANN code generator in a HDL language. So the designer does not need to know anything about ANNs and just use them.

Another future work is to implement more ANN types, including for example RBF and Kohonen networks

8. REFERENCES

[1] A.B. Smith, C.D. Jones, and E.F. Roberts, "Article Title," *Journal*, Publisher, Location, pp. 1-10, Date.

[2] Jones, C.D., A.B. Smith, and E.F. Roberts, *Book Title*, Publisher, Location, Date.

[1] Braga, A. P.; Carvalho, A. P. L. F.; Ludermir, T. B. *Redes Neurais Artificiais*, LTC, 2007.

[2] Omondi, A. R. ; Rajapakse, J. C. ; Bajger, M. FPGA Neurocomputers. In: Omondi, A. R.; Rajapakse, J. C. (eds) FPGA Implementations of Neural Networks. Springer-Verlag, 2006. p. 37-56.

[3] Lee, Y.; Ko, S. B. FPGA implementation of a face detector using neural networks, IEEE CCECE/CCGEI, 2006.

[4] Krips. M, Lammert. T, Kummert. A. FPGA implementation of a neural network for a real-time hand tracking system. In: Proceedings of the First IEEE International Workshop, 2002. p. 313-317.

[5] Azhar, M. A. H. B.; Dimond, K. R. Design of an FPGA Based Adaptive Neural Controller For Intelligent Robot Navigation. In: Proceedings of the Euromicro Symposium on Digital System Design, 2002.

[6] Zeidman, B. An Introduction to FPGA Design, Embedded Systems Conference, 1999.

[7] Bernard, G. FPNA: Concepts and Properties. In: Omondi, A. R.; Rajapakse, J. C. (eds) FPGA Implementations of Neural Networks. Springer-Verlag, 2006. p. 63-101.

[8] Canas, A.; et al FPGA Implementation of a Fully and Partially Connected MLP. In: Omondi, A. R.; Rajapakse, J. C. (eds) FPGA Implementations of Neural Net-works. Springer-Verlag, 2006. p. 271-296.

[9] Girau, B. FPNA: Applications and implementations. In: Omondi, A. R.; Raja-pakse, J. C. (eds) FPGA Implementations of Neural Networks. Springer-Verlag, 2006. p. 103-136.

[10] Girones, R. G.; Agundis, A. R. FPGA Implementation of Non-Linear Predictors. In: Omondi, A. R.; Rajapakse, J. C. (eds) FPGA Implementations of Neural Net-works. Springer-Verlag, 2006. p. 297-323.

[11] Zhang, M.; Vassiliadis, S.; Delgago–Frias, J.G. Sigmoid generators for neural computing using piecewise approximations, IEEE Trans. Comput., 1996, p. 1045–1049

[12] Amin, H.; Curtis, K.M.; Hayes–Gill, B.R. Piecewise linear approximation applied to nonlinear function of a neural network, IEEE Proc. Circuits - Devices Sys., 1997 p. 313–317

[13] Basterretxea, K.; Tarela, J. M.; Del Campo, I. Approximation of sigmoid func-tion and the derivative for hardware implementation of artificial neurons, IEEE Proc.-Circuits Devices Syst., Vol. 151, 2004.

[14] IEEE computer society: IEEE Standard 754 for Binary Floating-Point Arithme-tic, 1985.

[15] Applet da rede neural XOR, disponível em: http://delfin.unideb.hu/~bl0021/xordemo/xordemo.html, último acesso em 23/11/2008.

[16] Krips, M.; Lammert, T.; Kummert, A. FPGA Implementation of a Neural Net-work for a Real-Time Hand Tracking System. Proceedings of the First IEEE Inter-national Workshop on Electronic Design, 2002.