NeuroFPGA - Implementing Artificial Neural Networks on Programmable Logic Devices

Daniel Ferrer, Ramiro González, Roberto Fleitas, Julio Pérez Acle, Rafael Canetti
doger@fing.edu.uy, {ramg,robfleitas}@adinet.com.uy, {julio,canetti}@fing.edu.uy
Instituto de Ingeniería Eléctrica
Facultad de Ingeniería - UDELAR
Montevideo - Uruguay

Abstract

An FPGA implementation of a multilayer perceptron neural network is presented. The system is parameterized both in network related aspects (e.g.: number of layers and number of neurons in each layer) and implementation parameters (e.g.: word width, pre-scaling factors and number of available multipliers). This allows to use the design for different network realizations, or to try different area-speed trade-offs simply by recompiling the design. Fixed point arithmetic with pre-scaling configurable in a per layer basis was used. The system was tested on an ARC-PCI board from Altera™. Several examples from different application domains were implemented showing the flexibility and ease of use of the obtained circuit. Even with the rather old board used, an appreciable speed-up was obtained compared with a software-only implementation based on Matlab neural network toolbox.

1. Introduction

Hardware implementation of neural networks has been in use for a long time as a means to accelerate the calculations. Technology used ranks from VLSI (Hammerstrom [7]) to programmable logic (e.g.: Arroyo et al. [3], Cox et al. [5], Girau et al. [8]).

Most of the previous works are focused on the resolution of a particular problem with stiff network parameters (e.g.: Aguilera et al. [4]). In that case, each problem requires a new design effort to take into account differences in network parameters. In order to obtain a general solution suitable for a wide range of problems, one of the major difficulties lie in the design of the automata controlling the circuit.

In the work presented here the reprogramability of FPGAs is used to implement a whole family of multilayer perceptron neural networks modifying only a set of parameters in compilation time.

The system is parameterized both in network related aspects and implementation parameters. The first group of parameters (number of layers, neurons per layer, etc.) are selected based on the particular problem to be solved. The second kind of parameters are chosen in order to optimize the area usage and throughput according to device limitations. This allows to use the design with different network dimensions and different area-speed trade-offs by simply recompiling the design.

The activation function was synthesized as a piecewise linear function, where the slopes of the sections are negative powers of two. This function is also parametric, and provides a good approximation to the $\tanh(x)$ function. Another parameter allows to disable the activation function in the output layer, so the network output values are not bounded to the range of activation function output.

The obtained design is described in AHDL which assures its portability to other Altera programmable logic devices. The resulting circuit was synthesized using commercial Altera design tools and was tested on the ARC-PCI board for several network examples.

2. Mathematical model of neural networks

A multilayer perceptron consists of a set of units called neurons interconnected configuring a network. Each neuron has a set of inputs x_i, and one output y_j; each input is affected by a weight coefficient w_{ji}. The subscript ij refers to the input i in neuron j.

$$y_j = \varphi(v) = \varphi\Big(\sum_i w_{ji}x_i + b_j\Big) \qquad (1)$$

The computation made by each neuron is a sum of products affected by a nonlinear function, given by the equation 1. The function φ is a limited nonlinear function usually referred as activation function.

1530-1591/04 $20.00 (c) 2004 IEEE

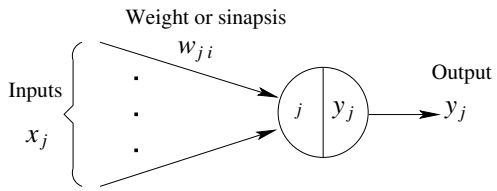

Figure 1. Mathematical model of an artificial neuron

Neurons are arranged in layers. The outputs of each layer's neurons are inputs of the neurons in the following layer. The outputs of the network are provided by neurons in the output layer.

3. Circuit Design

The implemented topology network is the "feed-forward", fully connected network.

In order to have a flexible and scalable neural network implementation, many parameters were defined: they arrange in two main groups. The first one is related to the architecture of the network: the number of layers (`ncapas`), the number of inputs (`nent`), the number of neurons in each layer (`nneu1`, `nneu2` and so on), and the option of bypassing the activation function in the output layer to obtain a linear output (`salida_lineal`). An example of the architecture parameters is shown in figure 2 for a three layer network.

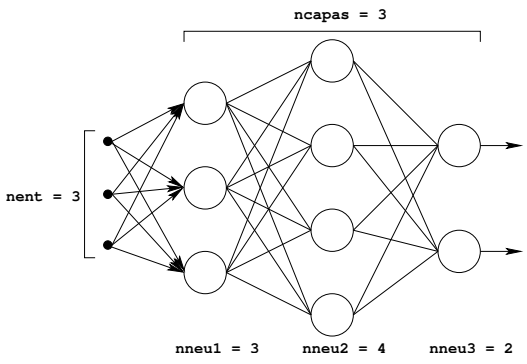

Figure 2. Example parameters for a 3 layer network

The second group of parameters is related to implementation restrictions: word width (`n`), internal word width (`l`), number of available multipliers (`mx_max / mult`), maximum pre-scaling factor (`esc_max`), pipeline deep (`pipe_deep`), clock frequency divider (`nclks`) and activation function slope (`p`).

Calculations are performed in fixed point arithmetic and the inputs and weights are numbers normalized to the range $[-1, 1)$ represented using two's complement fixed point words of $\mathtt{n}+1$ bits.

For a given a problem, the input and output patterns can always be scaled within the interval $[-1, 1)$. On the other hand a trained net may have weights or biases laying outside this interval. In that case an offline pre-scaling of weights and biases is performed, compensated by a post-scaling in the argument of the activation function provided by the circuit. This scaling is an integer number power of two, and must be constant for each layer.

The activation function synthesis is discussed later.

3.1. Architecture and Data Path

To obtain a general solution, even under area restrictions, the multipliers that fit in the chip are reused as many times as necessary to obtain the result of a complete neuron. The process is iterated for all the neurons of a layer and for all the layers, until obtaining the outputs of the complete network.

The biases and weights are used in the calculation for each input pattern. They are stored in the on-chip RAM memory. A RAM block (weights RAM) is associated to each multiplier, for storing the weights corresponding to all the synapsis computed by it. The weights are stored at the beginning of the operation and the RAM block is operated as a circular queue.

Analogously, another on-chip RAM block (biases RAM) is used to store the biases of all the network neurons.

The board has two RAM chips connected to independent buses (bus 1 and bus 2) which are used to store the inputs and outputs of the network respectively. To start the calculation corresponding to a set of input patterns, these patterns are loaded by the host in the RAM connected to bus 1.

Two other register blocks implemented on-chip (temporary storage REG 1 and REG 2 in figure 3) are used alternately to store neuron inputs and outputs respectively. At each layer computation, one of the blocks contains the layer's inputs. As the layer outputs are obtained, they are stored in the other block. For the next layer computation the recently obtained results are used as inputs and the block that initially had the inputs is used to store the new layer outputs. When the results of the output layer are obtained, they are written directly to external memory through bus 2, and simultaneously the next network inputs are read from external memory to a RAM block through bus 1.

The multipliers receive as inputs the data in the weights RAM and one of the temporary RAMs. To evaluate each neuron, in the initial pass the content of the biases RAM is

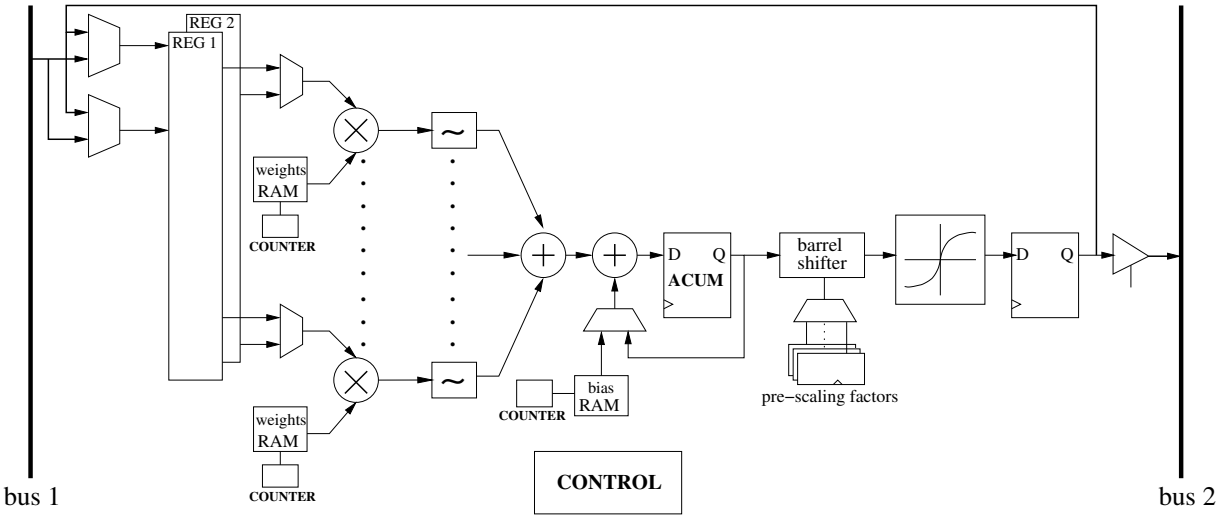

Figure 3. Circuit block diagram

added to the outputs of the multipliers and stored in an accumulator register. In case the neuron has more inputs than the available number of multipliers, additional passes are needed adding the accumulator register content instead of the bias.

When all the inputs were added the output of the neuron can be obtained from the activation function block connected to the accumulator.

The timing of this process is shown in figure 5 (section 3.3) for an example network with 3 layers.

The inputs and weights are represented in $\mathtt{n}+1$ bits fixed point arithmetic. When multiplied, the result grows to $2\mathtt{n}+1$ bits. The output of the multipliers can be truncated to $\mathtt{l}+1$ bits, with $\mathtt{n} \leq \mathtt{l} \leq 2\mathtt{n}$, being $\mathtt{l}$ a parameter.

Before evaluating the activation function, the previous optional scaling in weights and biases is compensated by shifting the same displacement for all the neurons in a layer. This is done using a barrel shifter.

As the combinatorial logic evaluating the sum of products is the critical path, it was provided with a configurable deep pipeline to reduce the maximum delay between registers.

3.2. Control unit

The neural network is mainly controlled by a state machine. Its simplified diagram is shown in the figure 4.

Once the FPGA device is configured, the circuit "borns" in an *idle* state, waiting a command signal to start the configuration of the network (state *initialize*) or to start the evaluation of a set of input patterns (state *calculus*). This commands are issued from the host by writing in reserved memory addresses (input signals: `start_inic` or `start_calc`).

When entering state *initialize*, the circuit requests access to bus 1, and once acknowledged, reads in sequence starting from address 0 (one word each time, one word by memory address) the weights, biases and scale values for each layer. It stores them in the weights RAM, biases RAM and dedicated registers respectively. Once this stage is completed, the circuit returns to the *idle* state.

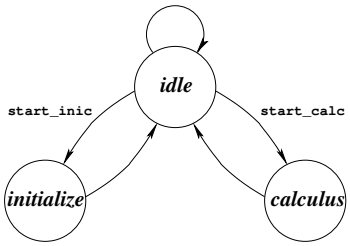

Figure 4. Simplified state diagram of the circuit

On the other hand, in the *calculus* state, the circuit requests both buses and when it gains them it begins to read from the memory connected to bus 1 the amount of input patterns to process (in the address 0) and the first of them. After this pattern is processed, the circuit starts the iteration process explained in the previous section. This iteration is repeated until the amount of sets stored in the memory is processed, returning again to the idle state.

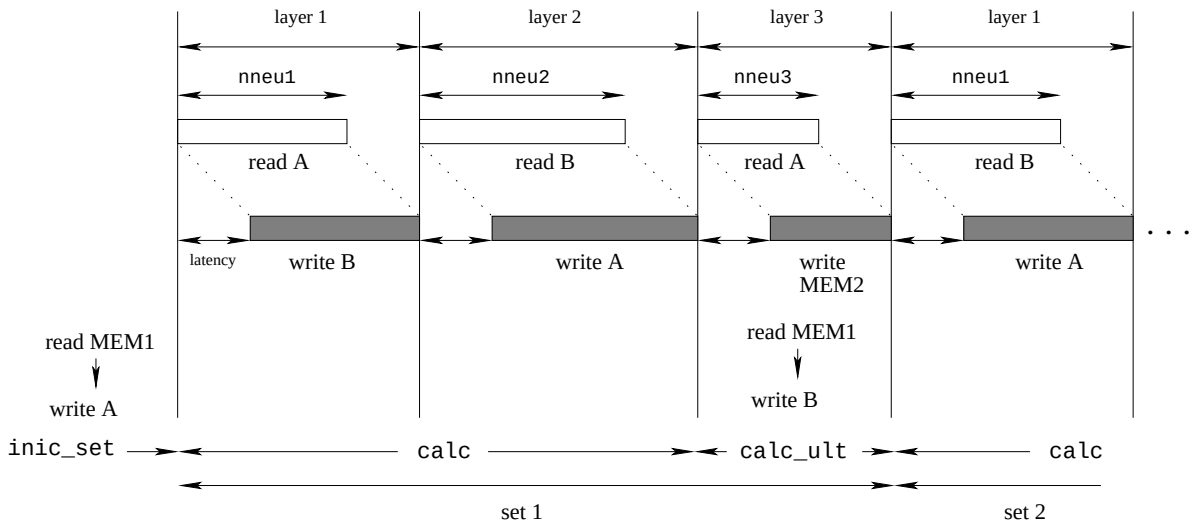

Figure 5. Simplified timing diagram of the circuit

3.3. Timing diagram

Figure 5 shows an example of the operation for a 3 layer network, having `nneu1`, `nneu2` and `nneu3` neurons in each layer, respectively. The *calculus* state in the simplified diagram in figure 4 is divided in 2 stages: `calc` and `calc_ult`. Layers 1 and 2 are processed in stage `calc` and the last layer in `calc_ult`. The two temporary RAMs described above are designated "A" and "B".

3.4. Activation function

An activation function with sigmoidal shape that approximates very well the following function was obtained:

$$\varphi(v,a) = \frac{1 - e^{-av}}{1 + e^{-av}} \qquad (2)$$

As in Aguilera et al. [4], it was implemented as a piecewise linear function. However, in the work presented here the slopes of each segment are negative and consecutive powers of 2 synthesized with a simple combinatorial circuit, optimizing input-output delay. All the operations are performed using barrel shifters and OR-AND logic gates.

This approximation is shown in figure 6.

4. Hardware platform

Several test networks and application examples were synthesized and tested to validate the design. An ARC-PCI board from Altera was used. This board plugs in a personal computer's PCI slot and has 3 EPF10K50 FPGA chips. Each FPGA has 2880 logic cells (50,000 gates) and 20Kbits of on-chip memory (EABs).

The board can also be equipped with up to 4 static memory banks, that can be accessed from the FPGA chips.

One of the chips is connected to the PCI bus and thus hosts the PCI interface and the memory buses arbitration logic. Previous designs from Oliver [9] and Bishop [10] were used for the interface chip allowing data transfer between the host PC and the memory banks.

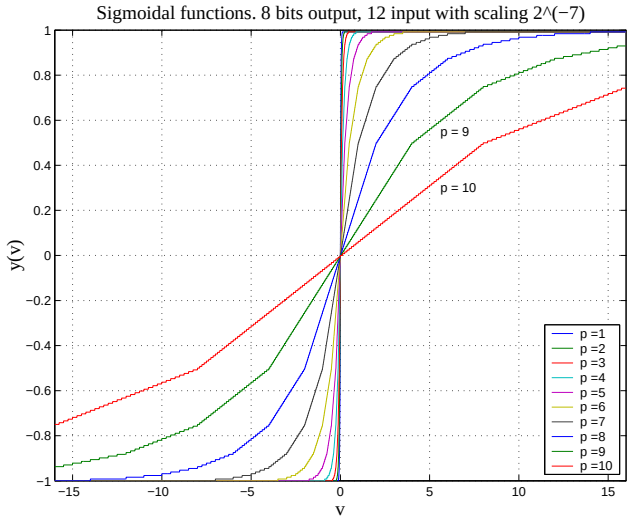

Figure 6. Family of sigmoidal functions obtained

One of the FPGA chips was used to accommodate the

neural network circuitry. Two memory banks were used in order to allow writing the results for the current set at the same time that the input values for the following set are read.

Once the target hardware is given, several bounds are imposed by the available memory and area. On one hand the available area limits the number of multipliers that can be synthesized on the chip for a given word width. This limitation can be avoided by repeatedly using the same hardware as explained above. The available external memory sets an upper bound to the number of input sets that can be processed without host intervention. The available on chip memory, on the other hand, limits the amount of weights and biases that can be stored and thus sets a bound to the neural network size.

Figure 7. ARC-PCI board

5. Example applications

In order to demonstrate the ease of use of the system, several example applications were selected from different areas. Furthermore, for a given platform there is a trade-off between the selected data word width (affecting the numeric precision) and the number of multipliers that fits in the available area, and consequently the processing speed. In each example, given the trained network description, several compilations were performed to select the fastest solution with an acceptable error. All this process was done in a very short time (within one working day for each example), showing the flexibility of the obtained environment.

The results for two applications that were compiled and ran on the ARC-PCI board are shown. The first one is a classic pattern classification problem, and the second one is a function approximation. Besides the applications tested on the ARC-PCI board, several networks were compiled for more recent chips and the resulting processing speed was evaluated.

5.1. Classification problem

The application consists in recognizing the five Spanish language vowel phonemes given a set of characteristics extracted from a sampled voice signal.

The architecture of the network is the following: 2 layers, 8 inputs, 6 neurons in the input layer and 5 neurons in the output layer, one activated for each vowel. The presence (or absence) of a vowel is coded as a '1' value (or '-1' value) in the corresponding output.

The starting point was a trained network with normalized patterns in the range $[-1, 1)$, using the hyperbolic tangent as activation function. The circuit was synthesized for different combinations of parameters. The obtained performance was 490 thousands sets per second (about 38 million products weight-input per second), using a clock of 16,67 MHz in a circuit with 8 multipliers of 8 bits. It correctly classified all the sets of inputs tested.

The problem was taken from a bigger application developed as a final project in a graduate course (Facciolo et al. [11]).

5.2. Function approximation problem

This problem was obtained from PROBEN1 (Prechelt [6]), and it consists in the forecast of energy consumption in a building. The goal is to predict the per hour consumption of electrical energy, hot water and cold water, as a function of the date, hour of the day, room temperature, ambient humidity, solar radiation and wind speed. The network has 14 inputs, 3 layers with 8, 8 and 3 neurons in the first, second and output layer respectively, and the activation function is bypassed in the output layer.

The network was trained in Matlab with a set of 2104 examples, using the tanh(x) activation function.

The circuit was synthesized for several parameter combinations. The selected configuration resulted in a performance of 231 Ksets/s or 46,3 Mproducts/s (Million weight-input products per second) for the same 16,67 MHz clock frequency and also using 8 multipliers of 8 bits. The numerical error was less than 0,77 percent of the full range.

6. Results

As a summary, the table 1 shows a performance comparison between the circuit synthesized in the ARC-PCI board and software-only solutions developed using the neuronal networks toolbox of Matlab, for the two applications mentioned above.

Although Matlab uses a different numerical representation, it is presented here because as a widely used simulation tool, is an easily understood reference.

	ARC-PCI	Matlab
Classification problem	490.3 Ksets/s 38.24 Mprod/seg	117 Ksets/s 9.13 Mprod/s
Functional approximation	231.5 Ksets/s 46.31 Mprod/seg	43 Ksets/s 8.60 Mprod/s

Table 1. Summary of performance obtained in ARC-PCI compared with Matlab

The circuit in both cases uses 8 multipliers of 8 bits each and a 16,67 MHz clock, resulting in an occupation of 85% and 91% of area of FLEX10K50 FPGA in each case. The performance obtained in Matlab was measured using the function *sim* from the neural networks toolbox (calculation made with all the precision of Matlab, 64 bits floating point), running Windows XP on an Athlon XP 1700+ cpu with 256MB of RAM.

Besides, the design was compiled for current APEX II, ACEX and Stratix Altera's technologies, which are bigger and faster than the ARC-PCI chips. The best results were for a Stratix chip where the synthesis tools reported an expected performance of **360 million products per second** for a big network using 48 multipliers of 16 bits each and a 35 MHz clock. Currently, the price of this chip is about USD 1200 in small quantities.

7. Conclusions

A flexible and scalable design was obtained for a whole family of multilayer perceptron neural networks. The circuit for each new application can easily be generated by setting the parameter values to match the particular network sizes and running the synthesis. Similarly for a particular network, different parameter combinations can easily be synthesized to select the best solution for a given target hardware.

The examples showed that very short design cycles can be achieved allowing to obtain a running application in a very short period since its conception.

The design was hardware validated on an FPGA board. Even with the rather old board used, an appreciable speed-up was obtained compared with a software-only implementation based on Matlab neural network toolbox.

8. Acknowledgements

We wish to thank Altera for providing the ARC-PCI board where this work was implemented, and also Juan Pablo Oliver from our University for kindly furnishing the PCI core.

References

[1] Haykin, Simon, *Neural networks: a comprehensive foundation*. Second edition, Prentice-Hall 1991, ISBN 0-13-273350-1.

[2] James A. Freeman/David M. Skapura, *Neural Networks - Algorithms, Applications, and Programming Techniques*. Addison-Wesley, 1992, ISBN 0-201-51376-5.

[3] Marco A. Arroyo León, Arnoldo Ruiz Castro, Raúl R. Leal Ascencio, An Artificial Neural Network on a Field Programmable Gate Array as a virtual sensor, *Proceedings of the International Symposium on Robotics and Automation - ISRA'98*, Saltillo, Coahuila, Mexico, 1998.

[4] Cuauhtemoc Aguilera Galicia, Raúl R. Leal Ascencio, A CPLD Implementation of an Artificial Neural Network for Instrumentation Applications, *Second International Workshop on design of Mixed-Mode Integrated and Applications*, Guanajuato, México, 1998.

[5] Charles E. Cox and W. Ekkehard Blanz. GANGLION - A Fast Field Programmable Gate Array Implementation of a Connectionist Classifier, *IEEE Journal of Solid-State Circuits*, 27(3):288-299, March 1992.

[6] Lutz Prechelt, *Proben1 - A Set of Neural Network Benchmark Problems and Benchmarking Rules*. Fakultät für Informatik, Universität Karlsruhe, Germany, 1994 Technical Report.

[7] D. Hammerstrom, *A VLSI Architecture for High-Performance, Low-Cost, On-chip Learning*. Proceedings of the International Joint Conference on Neural Networks. 1990

[8] B. Girau, A. Tisserand, *On-line Arithmetic based Reprogrammable Hardware Implementation of Multilayer Perceptron Back-Propagation*. IEEE Computer Society, 5th International Conference on Microelectronics for Neural Networks and Fuzzy Systems (MicroNeuro96). 1996

[9] Juan Pablo Oliver, Algoritmos en lógica programable, *Ongoing Master Thesis*, http://iie.fing.edu.uy/∼jpo

[10] William Bishop, The ARC-PCI board page, http://www.pads.uwaterloo.ca/∼wdbishop/arc-pci/

[11] Gabriele Facciolo/Diego Rother, *Reconocimiento de Voz*. Final project for course "Sistemas Neuro-Fuzzy", 2002. http://iie.fing.edu.uy/ense/asign/neurofuzzy/