ARCHITECTURE AND DESIGN METHODOLOGY OF THE RBF-DDA NEURAL NETWORK

Mourad Aberbour (student member), Habib Mehrez (member)

LIP6/ASIM laboratory
Université Pierre et Marie Curie
4, Place Jussieu, 75252 Paris Cedex 05, France
e-mail:mourad.aberbour@lip6.fr

ABSTRACT

This paper presents a novel implementation of the RBF-DDA (Radial Basis Functions - Dynamic Decay Adjustment) neural network. We will investigate the architectural design and the hardware implementation. The features of this design are the possibility of mapping the architecture on different standard cells libraries as well as FPGA's, and mainly its modularization. This is an example which reflects the new trend in design methodologies relying on re-use and IP blocks. Indeed, the architecture is implemented using intensively the module generator and macro-blocks concept. The design is thus readily reusable, flexible and upgradable. The architecture is parameterized in terms of the precision of the data processed by the network and in terms of the size of the network itself (number of neurons). The neural network is used in the classification of the image signatures which are extracted from gray-level images using the Gabor-Morlet wavelets and some data compression schemes. The RBF-DDA approach implements the Probabilistic Neural Network model. The activation function is a Gaussian which models the probability distribution of the categories to be differentiated. In the introduction we briefly describe the whole recognition system and we scan rapidly the hardware implementation experiences. The second section discusses the RBF-DDA algorithm and architecture. Then we present the architecture and the VLSI design methodology of the RBF network. We finally present some implementation results and conclude this paper.

1. INTRODUCTION

The study and design of the RBF-DDA neural network is a part of a bigger project on target recognition based on the Gabor wavelets and neural networks[1]. A pattern recognition system can be considered as a two stage device. The first stage is feature extraction and the second is classification. A certain number of feature vectors are extracted which represent the main characteristics of the gray level image. These vectors, after concatenation, are the signature of the object to be recognized. They are then fed to the Neural Network for classification.

Several VLSI implementations of neural networks have been attempted in the last years either by industry or academia. The proposed solutions to list a few of them range from neuro-accelerator platforms to digital or analog or even hybrid ASICs. Designers have concentrated their efforts for the design of a limited number of neural network categories. These are sigmoid neural networks, Radial Basis Functions (RBF) neural networks and the Kohonen self-organizing maps.

One of the disadvantages of multilayer perceptrons (MLP) is that the learning is quiet complex, heavy in calculations thereby requires longer time, and is hard to implement in an ASIC. MLP loses ground when a new class or even few new patterns are added, it requires that training be started from scratch (all patterns, learned or not, be presented to the network again and again).

The basic concept of RBF is that different patterns are stored while training. Classification is done by calculating the distances (Euclidean or Manhattan) between the input pattern and all the stored prototypes. These distances are then manipulated in order to classify the input pattern or vector. The classification of RBF is shown in figure 1. The advantages of RBF consist of simple and less time consuming and easy to implement training algorithm in an ASIC chip, cascadibility of several chips is easily achieved, and easy updating when a new class or category is added.

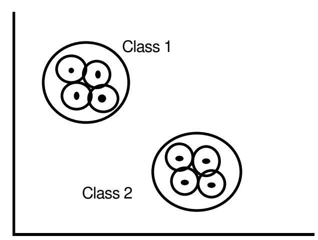

Figure 1. RBF classification in 2-D

The analog implementations of neural networks, like the Intel ETANN, the CAM of SGS-Thomson and the AT&T ANNA, present two main drawbacks. They support very low precisions (less than 8 bits) and also a low weight storage capacity.

Two main categories of neural networks have been implemented using the digital technology. First, the MLP feed-forward models are implemented in the NeuroLogix NLX-420, the MCE MT19003, the Micro Devices MD-1220, and the Phillips Lneuro-1 [8]. Most of them implement an off-chip memory for the weights, which presents an I/O bandwidth problem, or a very limited memory. Two known RBF implementations have been achieved by the industry. These are the IBM ZISC036 and the Intel NI1000. The ZISC implements an RBF-like neural network model which is a very simplified version of the original RBF specification. It offers many interesting features like easy cascadability and simplified utilization. The drawback of this chip is the limited dimension of the prototypes (64) and the simplification of the algorithm at the expense of less mathematical rigor.

The Intel NI1000 implements also the RBF algorithm. It uses a PNN model. But the activation functions are sharp peaks instead of Gaussians which make its interpolation capabilities limited. It also has the drawback of limited precision for the prototypes, 5-bits.

In this paper we present an implementation of the RBF DDA model complying fully with the mathematical formulation of the algorithm. And with a comfortable precision and length of the prototypes (256 16-bits components). We will also present an architecture for implementing the learning on-chip. The main feature lies in the design methodology which relies on the generator and macro-block concept. This makes this design portable on any standard cells library and even over FPGAs.

2. RBF-DDA (DYNAMIC DECAY ADJUSTMENT)

Most algorithms used to train RBF networks require a fixed architecture, i.e., a fixed number of neurons. The RCE algorithm and its probabilistic extension, the P-RCE, use a growing architecture which introduces new hidden neurons dynamically. The RBF DDA algorithm uses the constructive nature of the P-RCE algorithm together with independent adaptation of each prototype's decay factor. It is shown that networks trained with the DDA algorithm perform substantially better than common RBF networks[9].

There are three layers of neurons, first the input neurons, which simply distribute the input. The hidden neurons store the prototypes and the standard deviation 'σ'. They calculate the Euclidean distance 'dist', divide the 'dist' by its 'σ^2', and then find the exponential as "$R=e^{-dist/\sigma^2}$". The number of output neurons indicates the number of classes/categories we have. Each hidden neuron is connected to all of the output neurons by a certain weight 'A_i'. These weights are integers. The output neurons multiply the output 'R_i' of the hidden neurons with corresponding weight 'A_i' and accumulate the result ($\Sigma A_i R_i$).

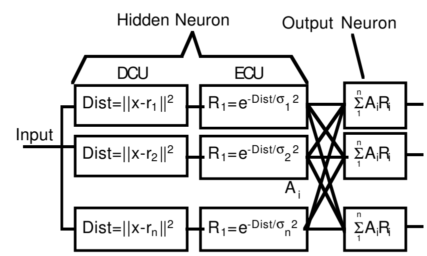

Figure 2. RBF-DDA Architecture

The training determines the number of hidden neurons, i.e which and how many input patterns are stored, their standard deviation, and the weight by which a hidden neuron is connected to the output neurons. An interesting point is that all the weights by which a hidden neuron is connected to output neurons are integers which eases the hardware implementation. For classification Winner-Takes-All (WTA) algorithm is used on these accumulated

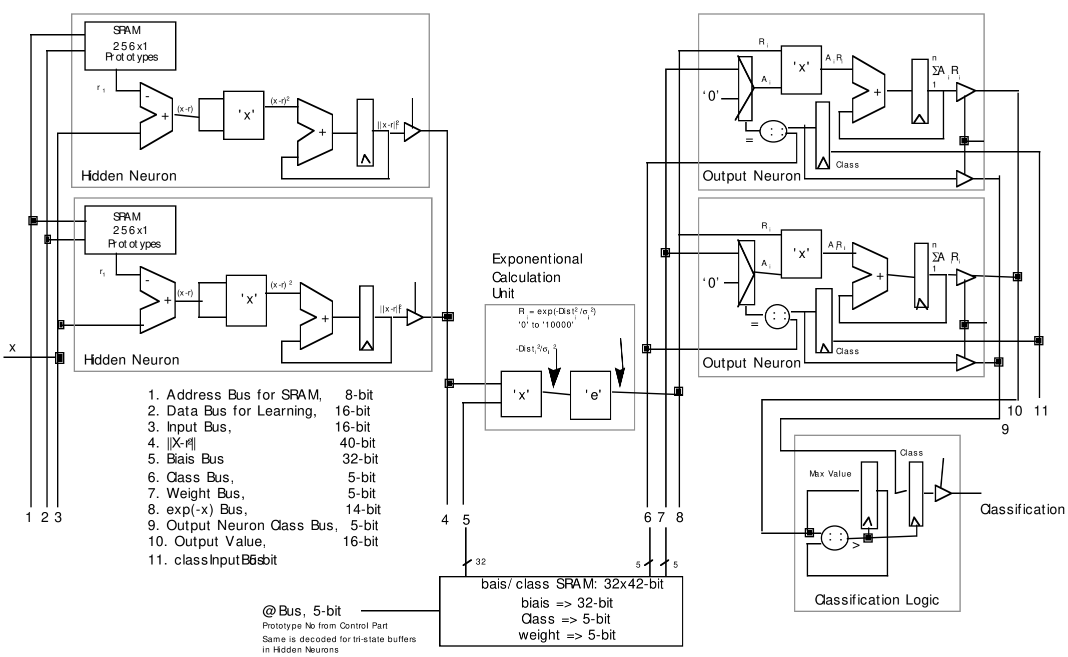

Figure 3. The Complete Chip; 'n' DCUs, one ECU & two Output Neurons

values. Figure 2 shows the RBF-DDA model. DCU & ECU mean Distance and Exponential Calculation Unit respectively.

3. THE HARDWARE IMPLEMENTATION

The architecture of the DCU, ECU and the output neuron of figure 2 is shown in figure 3. Each DCU has ‘m’ registers to store the ‘m’ components of the prototype. The first adder and the multiplier are 32-bit whereas the second adder is 64-bit. The value of ‘m’ is decided to be 256. There will be only one ECU for all the DCUs, as opposed to the figure 2. In fact we are forced to serialize the output ‘R_i’ in order to feed the output neurons because all the output neurons need ‘R_i’ from all the hidden neurons and the best it to connect all the outputs of “hidden neurons” (ECU) and the input of “output neurons” via a BUS. So, if we serialize at the output of DCU we will gain in surface of the chip as we will have only one ECU as compared to ‘n’ required for ‘n’ hidden neurons. See Figure 3.

The output neuron is a Multiply Accumulate architecture (MAC). Each output neuron has ‘n’ registers to store weights for ‘n’ prototypes.

3.1 Squaring Unit

The distance calculation unit consists of an accumulator along with a squaring unit. Using a multiplier for this operation is not a good compromise. We can reduce the surface area of the squaring unit by more than one third w.r.t the multiplier by using the simplifications implicit to the squaring operation. More over for accumulation, we can add the accumulated value along with the minterms of the squaring operation and in the end use an RSA adder.

3.2 Exponential Calculation Unit

The exp(-x) gives a value between ‘0’ and ‘1’. We will use fixed point so that the output neurons use integer units. Standard deviation can be any value between 10^{-10} and 10^{+10}, so we code the $1/\sigma^2$ as “16.16-bit”. Thus the effective range is between $1.5\text{x}10^{-5}$ and $6.55\text{x}10^{4}$ which is acceptable. If we use 16-bit components for the prototypes, then we will be requiring a 32x32-bit multiplier and the result will be on 64 bits.

Now we have to use these 64-bits to calculate the exponential. We think that four digit precision after the decimal will be sufficient, that is 14-bits will suffice, ‘1.0000’ coded as ‘10000’. So now we can base our output neuron with 16-bit input. There are four ways to do it. First use a Look-Up table. We can easily code e^{-x} in a 256 x 16-bit table. With four digit precision the value $e^{10.0}$ is zero. In this case we will use only 4 bits before decimal and 8 bits after decimal out of the 64 bits available and the result will be 16-bit.

The second method is to use the exp() algorithm. The following algorithm proposed in Muller [3] is for the values between 0 and 1.56. It needs many iterations to achieve a meaningful precision.

```
float expo( float t)   /* 0<=t<=1.56 */
{           int i;
            float x=0, ex=1, u;
            for(i=0; i<N,i++)
            {   u=x+log(1+2^-i);
                if( u<= t) { x=u;  ex=ex+ex*2^-i; }
            }
}
```

The third method is to approximate it as a parabola and calculate the value as done by M. Hervieu [4]. Fourth is to approximate exp() as a straight line as done by Robert Slaviero [5]. All the four implementations have been studied and verified by simulation. Finally the one offering best performance and error compromise was the LUT implementation.

4. VLSI IMPLEMENTATION METHODOLOGY

A set of generators has been developed separately during the design of fully parameterized libraries of complex arithmetic and DSP operators (fixed and floating point) [12]. This set of generators are intensively used in the design of the RBF-DDA. The macro-blocks generated are surface and delay optimized. These generators are designed under the GenOptim environment.

4.1 GenOptim CAD environment

GenOptim allows the designer to concentrate exclusively on the algorithm to be realized. Furthermore, it makes it possible to undergo modifications without any intervention on the source code of the generator. Consequently, the developer of generators is never bothered by the electrical, placement and timing constraints. GenOptim allows the user to quickly design efficient macro-cell generators that do not rely on particular target technologies. These macro-functions (shifters, multiplexers, priority encoders, adders, multipliers,...) are very common in the VLSI design world and may be used in many different contexts.

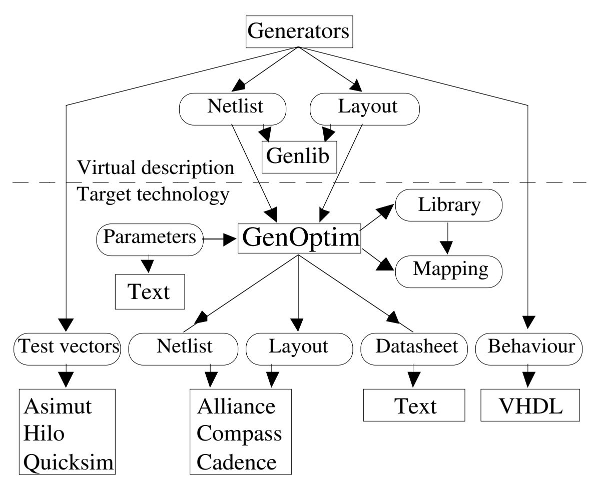

Figure 4. GenOptim description.

GenOptim uses a set of C functions which allow the user to describe the netlist, the layout, the test vectors and the behavioral views. As depicted in figure. 6, the circuit generation starts with a virtual description (netlist and layout of virtual cells) of the circuit. Then, GenOptim maps the virtual description to the target library and optimizes the circuit. A successfully fabricated image convolution chip following this methodology has been reported in [11].

4.2 Generators description

From simple parameters (name of the circuit and the number of bits), the generators create automatically a netlist view, a placement view, a behavioral view, test vectors and finally a file summarizing all the circuit characteristics. The generators are built up from two distinct parts. One proper to each generator and describes logically the algorithm to be realized using virtual cells. The other is common to all generators, called GenOptim that maps the virtual cells on the target library and solves automatically the electrical and placement problems by analyzing the circuit and the logic cells library used.

4.3 Performance & Surface Estimation

The design using the macro-block generator concept allows easy and large parameterization of the chip. The parameters are: precision and length of the prototypes, number of hidden neurons and output neurons to integrate as well as many other fine tuning parameters. We give performance examples for 256 components with 32 bits and 16 bits precisions and on different target technologies. Each hidden neuron will store 256 components as a RAM rather than bulky registers. We used the “RAGE”, the RAM Generator from Alliance 3.0 [7] to generate the RAM. A 256x32 RAM will require 2.46 mm^2 (in a 0.7 µm technology). The table below shows the surface area of various units used in the DCU for a 0.7 µm technology for 16 bits precision. With a 32 bits data precision we can achieve 9 hidden neurons in a 100 mm^2 chip. But 9 hidden neurons (DCU) in one chip is not enough. We should have about 30 to 50 DCU in one chip otherwise we will be requiring a large number of chips . If we consider 16-bit components of the prototypes, then we can accommodate 24 DCU in 100 mm^2 on a 0.7 micron technology.

Unit	Area (0.7µm)	Time
Adder (32-bit)	0.21 mm^2	6.9 ns
Multiplier (32-bit)	1.70 mm^2	27.0 ns
Adder (64-bit)	0.40 mm^2	8.9 ns
Total	2.31 mm^2	42.7 ns

It is expected that we will be able to integrate about 20% more DCUs if we use a squaring unit and an adder combined using redundant arithmetic. Since 0.5 µm, 3(plus)-layer of metal technology is now easily available, we can have about 25 to 30 DCUs for 32-bit components and more than 50 DCUs for 16-bit components.

If we have 'm' components and 'n' prototypes, then we require 'm' cycles to calculate the distance, and 'n' cycles for the exponential unit and output neurons ($\Sigma A_i R_i$), plus few more to find out the winner. Thus for 400 prototypes and 240 components, the classification would take about 1000 cycles. At 40 MHz it is 25 µs, sufficient for 1000 classifications/sec. We expect to achieve very enhanced performance using a denser technology (0.25 micron). The design with the macro-blocks generators concept makes this possible. The design is completely independent of the target cells library. Only the new parameters of the new library need to be provided for the GenOptim tool.

5. CONCLUSION

The RBF neural network with the DDA learning has been shown to offer better performance. It offers many advantages attractive for a VLSI implementation. These are: modularization, regularity, and fast learning. The implementation of the RBF-DDA network complying to the rigorous mathematical formulations is addressed. A major issue in this design is the use of advanced arithmetic aspects and the implementation following the module generation concept. This is of great interest as it allows the parameterization, portability and reuse of the design, and suits well the nowadays design trend relying on IP blocks. This makes our design upgradable. And in the future we can meet stronger constraints and achieve higher performance.

6. REFERENCES

[1] Clark & Al., "*Gabor Transforms and Neural Networks for Automatic Target Recognition*", Work shop on Neural Networks, 1991

[2] Reinhold W., "*Design and FPGA-Implementation of a Neural Network*", Institute for Technical Informatics; ICSPAT96 proceedings, vol 2, Boston Oct96

[3] Muller J.M., "*Arithmetique des ordinateurs, operation set fonctions elementaires*", Masson, 1989

[4] Hervieu M., "*Etude Architecturale d'un système de reconnaissance en ligne de caratères manuscrits*", LIP6 report, 3rd December, 1996

[5] Slaviero R., "*Simplifications to artificial neural network backpropagation training and operation implementation using DSPs*", ICSPAT96 proceedings, vol 2, Boston Oct96

[6] SNNS, "*Stuttgart Neural Network Simulator*", University of Stutttgart, Germany

[7] Greiner A., Pecheux F., "*ALLIANCE: A complete Set of CAD Tools for teaching VLSI Design*", proc. Third EuroChip Workshop, 1992

[8] IBM ZISC036 data sheet, "*www.ibm.com*"

[9] Berthold & al, "*Boosting the performance of RBF networks with dynamic decay adjustment*", Advances in Neural Information Processing Systems, vol 7, 1995

[10] Aberbour M. & al, "*A real time Image convolver*", ICSPAT96, vol 2, Boston Oct, 1996

[11] Lindsey C.S., Lindblad T., "*Review of Hardware Neural Networks: A user's perspective*", Third work shop on neural networks, Italy 1994

[12] Aberbour M. & al, "A fully parametrized IEEE floating point operators library for use in digital signal processors", ICSPAT95, Boston Oct, 1996.