Digital Hardware Architectures of Kohonen's Self Organizing Feature Maps with Exponential Neighboring Function

Jorge Peña
Advanced Learning and Research Institute
ALaRI
Università della Svizzera Italiana
USI, Switzerland
jorge.pena@alari.ch

Mauricio Vanegas
Andrés Valencia
Microelectronics Research Group
Universidad Pontificia Bolivariana
Medellín, Colombia
mauricio.vanegas@upb.edu.co

Abstract

Kohonen maps are self-organizing neural networks that categorize input data, capturing its topology and probability distribution. Efficient hardware implementations of such maps require the definition of a certain number of simplifications to the original algorithm. In particular, multiplications have to be avoided by means of choices in the distance metric, the neighborhood function and the set of learning parameter values. In this paper, one-dimensional and bi-dimensional Kohonen maps with exponential neighboring function and Cityblock and Chessboard norms are defined, and their hardware architecture is presented. VHDL simulations and synthesis on an FPGA of the proposed architectures demonstrate both satisfactory functionality and feasibility.

1 Introduction

Artificial neural networks are parallel computational models comprised of densely interconnected adaptive processing units (neurons), able to learn a static map (supervised learning) or to classify and categorize the input space (unsupervised learning) from an input data [3]. A Kohonen's self-organizing feature map (Kohonen's SOFM) is an artificial neural network of unsupervised learning that captures the topology and probability distribution of input data [5].

Most of the neural networks applications have used simulations on conventional single-processor machines. The possibility of parallel processing and short operation times has encouraged implementations of hardware neural networks [6], [8], [10], [12]. Kohonen's SOFM has not been the exception, and there are several implementations in analog [7] as well as digital circuits [2] [9]. Generally, digital implementations have been more successful because of their lesser vulnerability to noise and their higher scale of integration when is compared with their analog counterparts.

In this paper we proposed digital, hardware-friendly architectures for one-dimensional and two-dimensional Kohonen maps. The main novelty with respect to existing approaches is the definition of an exponential neighboring function, which properly balances flexibility and simplicity.

The paper is organized as follows: section II introduces original Kohonen's SOFMs. Section III presents the proposed, simplified algorithm, suitable for hardware implementation. Section IV describes the digital architectures of both one-dimensional and two-dimensional maps. Section V presents the experimental settings, simulations, results and discussions. Finally, section VI states conclusions and suggests future work.

2 Kohonen's Self Organizing Feature Maps

A Kohonen's self-organizing feature map (SOFM) is an unsupervised learning neural net that captures the topology and probability distribution of input data [3], [5]. Its architecture consists of an array (a map) of units or neurons with a fixed position $\mathbf{R}_j$ within the map and a variable n-dimensional weigth $\mathbf{W}_j$, where n is the dimensionality of input patterns.

The weigths of the neurons are updated whenever a new input pattern is presented to the net, making them more similar to the current pattern. How much the weight of a single neuron will be updated depends on the proximity of the neuron to the winner unit in the map, identified by:

$$j^* = \arg\min_j D\left(\mathbf{W}_j, \mathbf{P}^k\right), \tag{1}$$

where $\mathbf{W}_j$ is the weight of the jth unit, $\mathbf{P}^k$ is the input

1-4244-0690-0/06/$20.00 ©2006 IEEE.

pattern presented at time step k, and $D(\mathbf{x}, \mathbf{y})$ is a distance function. After a winner take all process has identified the winner unit, the weight of the jth unit is updated according to

$$\begin{aligned} \mathbf{W}_j &= \mathbf{W}_j + \Delta\mathbf{W}_j & (2) \\ &= \mathbf{W}_j + \rho \cdot \Phi\left(\mathbf{R}_j, \mathbf{R}_{j^*}\right) \cdot \left(\mathbf{P}^k - \mathbf{W}_j\right) \ , & (3) \end{aligned}$$

where ρ is the learning rate and $\Phi(\mathbf{R}_j, \mathbf{R}_{j^*})$ is a neighboring function that (a) is normally symmetric, (b) returns values close to one for $\mathbf{R}_j$ close to $\mathbf{R}_{j^*}$, and (c) is monotonically decreasing with the metric $D\left(\mathbf{R}_j, \mathbf{R}_{j^*}\right)$.

At least two choices for the neighboring function are commmon in the literature. The first one is a *gaussian neighboring function*:

$$\Phi\left(\mathbf{R}_j, \mathbf{R}_{j^*}\right) = \exp\left(-\frac{D\left(\mathbf{R}_j, \mathbf{R}_{j^*}\right)}{2\sigma^2}\right) \ . \qquad (4)$$

The second one is what we call in this paper a *step neighboring function*:

$$\Phi\left(\mathbf{R}_j, \mathbf{R}_{j^*}\right) = \begin{cases} 1 & \text{if } D\left(\mathbf{R}_j, \mathbf{R}_{j^*}\right) \leq D_N \\ 0 & \text{otherwise} \end{cases} \ . \qquad (5)$$

The width of the neighborhood is controlled by the variance σ in the gaussian function of Eq. 4 or by the size of the neighborhood D_N in the step function of Eq. 5. In order to assure convergence of the unsupervised learning process, both the learning rate and the neighborhood width are decreased during the algorithm execution.

3 The Proposed Kohonen's SOFM

Typical software simulations of Kohonen's SOFMs use the euclidean metric to determine the winner unit and the neighborhood values:

$$D(\mathbf{x}, \mathbf{y}) = \sqrt{\sum_i \left(x_i - y_i\right)^2} \qquad (6)$$

A hardware implementation of this metric is costly in terms of silicon area due to the square root and the required multiplications. Similar problems arise with certain neighboring functions, such as the gaussian of Eq. 4. In order to achieve an efficient digital design, a number of choices regarding the distance metric and the neighborhood function, and certain simplifications regarding the set of possible values of learning step ρ, have to be made.

3.1 Distance Metric

Euclidean metric is one particular type of a *Minkowski metric* L_m:

$$L_m(\mathbf{x}, \mathbf{y}) = \sqrt[m]{\sum_i \left|x_i - y_i\right|^m} \ , \qquad (7)$$

with $m = 2$. Minkowski L_1 norm (aka City-block, Manhattan or Taxicab metric) and Minkowsi L_∞ norm (aka Chessboard metric) are particularly well suited for hardware implementations, due to the absence of roots and multiplications.

The Cityblock norm is given by

$$L_1(\mathbf{x}, \mathbf{y}) = \sum_i \left|x_i - y_i\right| \ , \qquad (8)$$

and the Chessboard norm is defined by

$$L_\infty(\mathbf{x}, \mathbf{y}) = \max_i \left|x_i - y_i\right| \ . \qquad (9)$$

The changes in the metric space introduced by using one of these norms instead of the euclidean norm can be intuitively grasped by looking at the way a circle is transformed under the change of norm. As it is shown in Fig. 1 circles degenerate into diamonds and squares. In a n-dimensional space, hyper-spheres degenerate into hyper-diamonds and hyper-cubes.

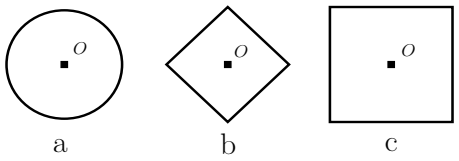

Figure 1. Graphical (euclidean) representations of circles (set of points in a plane that are equidistant from a given point O) under: (a) Euclidean Norm (Circle), (b) Cityblock Norm (Diamond), (c) Chessboard Norm (Square)

The Cityblock norm has normally been used in digital hardware implementations of Kohonen nets because of the easiness of adders implementation [9].

3.2 Neighboring Function

The gaussian neighboring function of Eq. 4 allows a different value of $\Phi\left(\mathbf{R}_j, \mathbf{R}_{j^*}\right)$ for each different $D\left(\mathbf{R}_j, \mathbf{R}_{j^*}\right)$ in the map. Furthermore, the neighborhood value of neurons at the same distance of the winner unit is decreased continuously by means of decreases in σ. Nevertheless,

the function is badly suited for hardware implementation due to the need of dividers and exponential blocks. On the other hand, the step neighboring function of Eq. 5 is simple enough to be used in hardware implementations [1], [9]. The downside of this neighboring function is its limited flexibility, given the fact that only binary neighborhood values can be returned.

A middle point between the two choices is proposed in this paper defining an *exponential neighboring function*:

$$\Phi\left(\mathbf{R}_j, \mathbf{R}_{j^*}\right) = \begin{cases} 1 & \text{if } j = j^* \\ \left(\frac{1}{2}\right)^{D\left(\mathbf{R}_j, \mathbf{R}_{j^*}\right)+\beta} & \text{if } j \neq j^* \end{cases}, \quad (10)$$

where $D\left(\mathbf{R}_j, \mathbf{R}_{j^*}\right)$ is the distance between the jth neuron and the winner neuron and β is a *width parameter* that controls the width of the neighborhood as σ or D_N did in the previously presented neighboring functions (Fig. 2). This neighboring function returns a different neighborhood value for each different distance between a losing neuron and the winning one, but keeps a simplicity that allows an efficient hardware implementation, as will be discussed later.

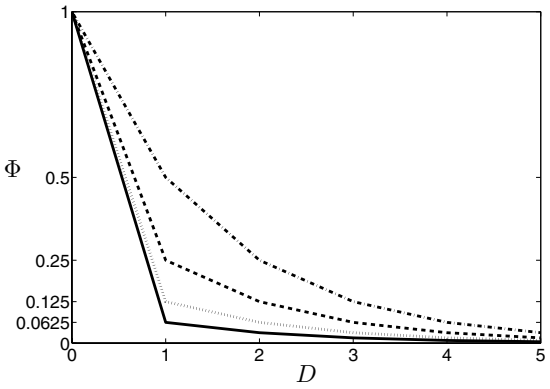

Figure 2. Exponential neighboring function with $\beta = 0$ (*dash-dot line*), $\beta = 1$ (*dashed line*), $\beta = 2$ (*dotted line*) and $\beta = 3$ (*solid line*)

3.3 Learning Rate Values

We complete the simplifications needed for our design restricting the possible values of ρ to the discrete values given by

$$\rho = \left(\frac{1}{2}\right)^{\alpha}, \quad \alpha = 0, 1, 2, \ldots \quad (11)$$

The decrease of ρ during learning can be achieved by incrementing α in a proper way.

3.4 Weight Update Formula

With the proposed simplifications, the update formula presented in Eq. 2 can be now written again as

$$\Delta\mathbf{W}_j = \begin{cases} \left(\frac{1}{2}\right)^{\alpha}\left(\mathbf{P}^k - \mathbf{W}_j\right) & \text{if } j = j^* \\ \left(\frac{1}{2}\right)^{\alpha+D\left(\mathbf{R}_j, \mathbf{R}_{j^*}\right)+\beta}\left(\mathbf{P}^k - \mathbf{W}_j\right) & \text{if } j \neq j^* \end{cases} \quad (12)$$

Hence, $\Delta\mathbf{W}_j$ is simply obtained by multiplying $\left(\mathbf{P}^k - \mathbf{W}_j\right)$ by a power of $1/2$, which can be easily done by shifting right $\left(\mathbf{P}^k - \mathbf{W}_j\right)$ the proper amount of bits.

4 Hardware Design

In order to check the suitability of the proposed model, both a one-dimensional and a two-dimensional Kohonen's SOFM digital architectures were designed by using a FPGA Spartan-3 of Xilinx Corp. and simulated by using Modelsim XE II 5.7g of Mentor Graphics [**?**]modelsim). To keep the designs as simple (yet interesting) as possible, we consider two-dimensional data with 8-bit resolution. Further generalization to n-dimensional data with the desired resolution should be easy to construct from this work.

The architecture of the one-dimensional map will be fully described. The architecture of the two-dimensional map is achieved introducing little changes on one-dimensional architecture.

4.1 One-dimensional Map

A 10-unit one-dimensional map like the one shown in Fig. 3 is considered.

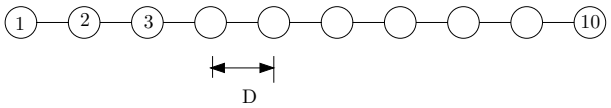

Figure 3. One-dimensional array considered for the design. Distance d between units is assumed to be one

The net is designed to work in parallel, as much as possible, to optimize execution speed. Each neuron is defined as a building block that processes data in an independently fashion. The proper behavior of the net, as an aggregation of these building blocks, is guaranteed by two "global" control blocks, namely *winner take all block* and *parameter scheduler*. The neuron architecture and the control blocks will now be briefly discussed.

4.1.1 The Neuron

A Kohonen's neuron is composed of a *distance calculation block* and a *weight update block* (Fig. 4). Each neuron includes four 8-bit and three 4-bit adders, one comparator (currently, the comparison $= 0$ is trivial) and two shifters. Though the shifters could be implemented with sequential machines (e.g., shift registers), they were actually implemented with combinatorial elements so as to optimize execution speed. Notice the absence of multipliers into the design. Notice also that, in the case of a one-dimensional map $\mathbf{R}_j = j$ and $D(\mathbf{R}_j, \mathbf{R}_{j^*}) = |j - j^*|$.

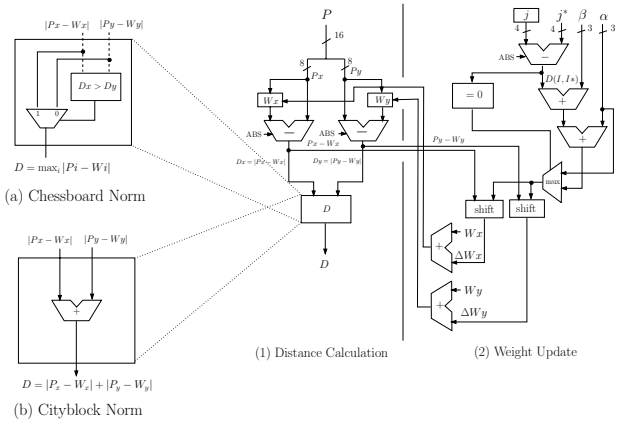

Figure 4. Neuron's architecture

In order to assure a proper synchronization of the digital neuron, the distance calculation block operates on falling edges and the weight updating block on rising edges of a clock signal, as shown in Fig. 5.

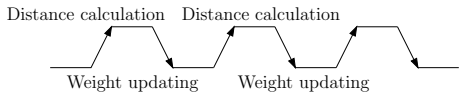

Figure 5. Neuron's synchronization

4.1.2 The Winner Take All Block

The winner take all block receives the distance D_j between each neuron j and input pattern and delivers the index j^* of the winner unit. This combinatorial block is composed of comparators and multiplexors arranged in a cascade configuration (Fig. 6).

4.1.3 The Parameter Scheduler

The parameter scheduler updates the learning rate parameter α and the neighborhood width parameter β incrementing them from an initial value of 0 to a final value determined beforehand. Basically, this block consists of two counters

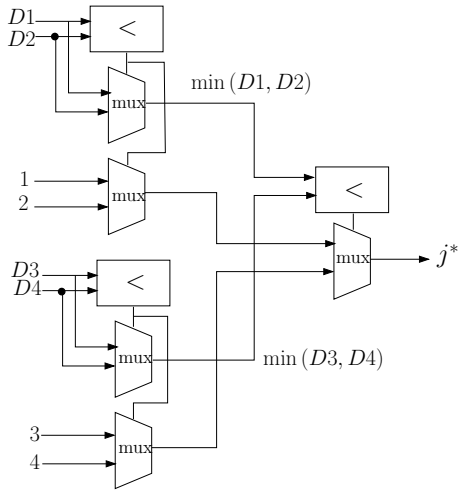

Figure 6. Winner take all block. For the sake of simplicity, a 4-neuron map is assumed

(one for each parameter), registers and a combinatorial logic to determine the set of time steps at which the parameters will be incremented.

4.2 Two-dimensional Map

A 25-unit two-dimensional map like the one shown in Fig. 7 is considered.

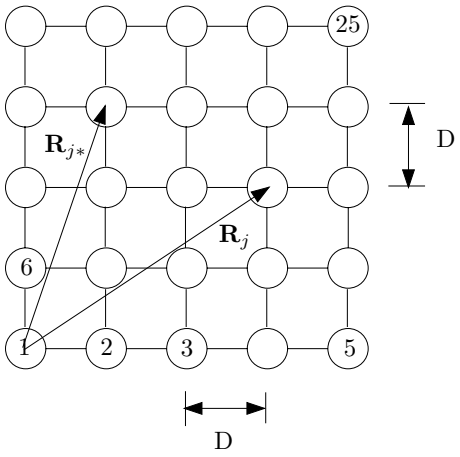

Figure 7. Two-dimensional array considered for the design. Distance d between units is assumed to be one

The two-dimensional map architecture is quite similar to that of the one-dimensional map. The winner take all block is essentially the same, but instead of propagating the

j values, it propagates the vertical and horizontal indexes of the neurons into the map (that is to say, $\mathbf{R}_j$), until the indexes of the winning neuron (that is to say, $\mathbf{R}_{j^*}$) have been identified. In the weight update block of the neuron (Fig. 4), the distance between the neuron and the winning neuron must be calculated using the specified norm, and not just as the absolute value of the diference of the j indexes.

5 Experimental Setup and Results

Both a one-dimensional and a two-dimensional Kohonen map were described in VHDL, simulated for simple clustering tasks and then synthesized on FPGAs.

5.1 Behavioral Simulation

5.1.1 One-dimensional, 10-unit, Chessboard norm Kohonen Map

A one-dimensional, 10-unit Kohonen map with Chessboard norm and exponential neighboring function was described in VHDL and then simulated for a simple clustering task. The input data consisted of 1000 points randomly generated in the xy plane between two concentric circles of center 125 and radii 70 and 100. Points between angles of 305 and 325 degrees were not included in the training set.

The net was trained for 18000 iterations (time steps). The parameters α and β were both initialized at zero, while the neuron weights were initialized at $(125, 125)$ for each one. The parameter α was incremented by one at time steps 4000 and 8000, and the parameter β at time steps 2000, 4000 and 6000. The scheduling is better shown in Fig. 8.

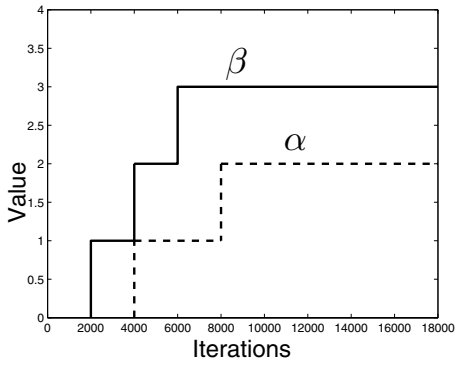

Figure 8. Parameter Schedule

Assuming a constant winning neuron during the whole execution (e.g, $j^* = 5$), the mentioned scheduling for the parameters α and β gives rise to the particular pattern of (normalized) weight changes in the map shown in Fig. 9. Observe the different values of the magnitude of the changes. This diversity is an advantage of the proposed exponentional neighborhood function over the traditional step function.

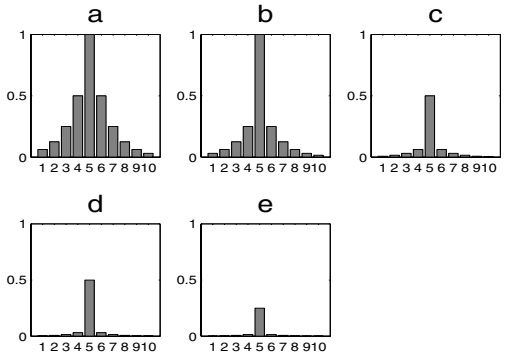

Figure 9. Normalized change in the weights $\left(\frac{\Delta \mathbf{W}_j}{\mathbf{P}^k - \mathbf{W}_j}\right)$ for a constant winning unit ($j^* = 5$) and different phases of the execution: (a) Iterations 0-2000 ($\alpha = \beta = 0$), (b) Iterations 2000-4000 ($\alpha = 0$, $\beta = 1$), (c) Iterations 4000-6000 ($\alpha = 1$, $\beta = 2$), (d) Iterations 6000-8000 ($\alpha = 1$, $\beta = 3$), (e) Iterations 8000-18000 ($\alpha = 2$, $\beta = 3$)

Fig. 10 shows graphs of the input data and the weight vectors for different phases of the algorithm. It can be seen how the units arrange themselves so as to follow the probability distribution of input vectors. Furthermore, the weight vectors are finally ordered according to their mutual similarity, so that neurons close to each other in the linear array correspond to neurons with close weights in the input pattern space, as must be the case of a well trained Kohonen net.

The histogram of Fig. 11 shows for each neuron the number of times it became the winner unit of the net. At the end of the run, each neuron has almost the same chance to win, which assures that the neurons have covered the input space following the probability distribution of the input data. Observe how the heights of the bars are close to the ideal theoretical value of 100 (1000 input patterns over 10 neurons).

A similar network, but with a Cityblock distance calculation block was also described in VHDL and simulated with the same input data and the same scheduling for α and β. The results were very similar to the ones obtained by the first network (with Chessboard norm).

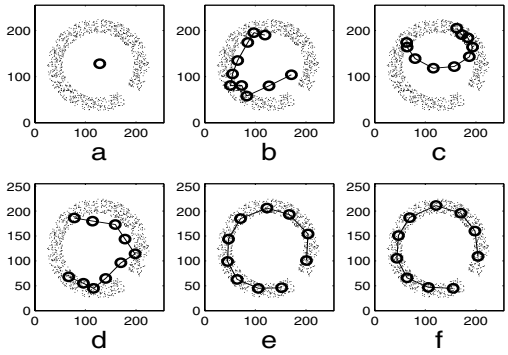

Figure 10. Input data for a simple clustering task (*small dots*) and weight vectors (*circles*) in the input space: (a) Iteration 0, (b) Iteration 1000, (c) Iteration 2000, (d) Iteration 3000, (e) Iteration 12000, and (f) Iteration 18000

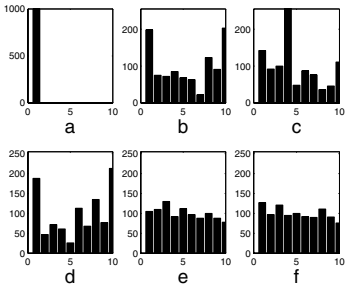

Figure 11. Histogram of winning events: (a) Iteration 0, (b) Iteration 1000, (c) Iteration 2000, (d) Iteration 3000, (e) Iteration 12000, (f) Iteration 18000

5.1.2 Two-dimensional, 25-unit, Cityblock norm Kohonen Map

A two-dimensional, 25-unit Kohonen map with Cityblock norm and exponential neighboring function was described in VHDL and simulated using the Modelsim Simulator XE II 5.7g of Mentor Graphics. The input data consisted of 1000 points randomly generated in the xy plane in the interior of a circle of center 125 and radii 100. The net was trained for 18000 iterations.

The scheduling of the parameters α and β was carried out as it was in the case of the one-dimensional map (Fig. 8). Fig. 12 shows the normalized change in the neuron weights according to the followed schedule.

Fig. 13 shows graphs of the input data and the weight vectors for different phases of the algorithm. Again, the

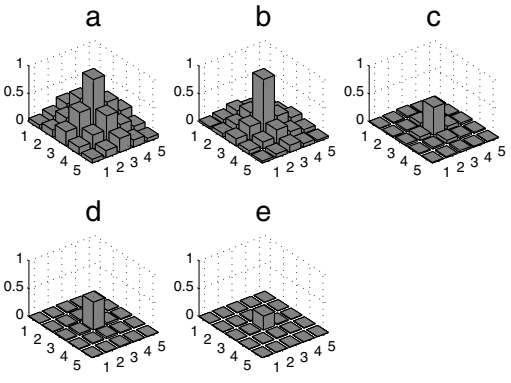

Figure 12. Normalized change in the weights $\left(\frac{\triangle \mathbf{W}_j}{\mathbf{P}^k - \mathbf{W}_j}\right)$ for a constant winning unit ($j^* = 13$) and different phases of the run: (a) Iterations 0-2000 ($\alpha = \beta = 0$), (b) Iterations 2000-4000 ($\alpha = 0$, $\beta = 1$), (c) Iterations 4000-6000 ($\alpha = 1$, $\beta = 2$), (d) Iterations 6000-8000 ($\alpha = 1$, $\beta = 3$), (e) Iterations 8000-18000 ($\alpha = 2$, $\beta = 3$)

neurons spread across the input space following the statistical distribution of the input data, and according to the topological relations in the map.

The histogram of Fig. 14 shows for each neuron the number of times it became the winner unit of the net. At the end of the execution, each neuron has almost the same chance to win, which again assures that the neurons have covered the input space following the probability distribution of the input data. The heights of the bars are close to the ideal theoretical value of 40 (1000 input patterns over 25 neurons).

As in the case of the one-dimensional map, a second network, identical to the one described but with the other proopsed metric (now the Chessboard metric) was also coded in VHDL and simulated following the same experimental settings. The results were very close to the ones obtained with the other norm.

5.2 Synthesis

The four Kohonen maps (the one-dimensional and two-dimensional maps implemented with both Cityblock and Chessboard metrics) were synthesized on an FPGA in order to determine the suitability of a real implementation of the proposed hardware. An FPGA is an array of logic cells whose functionality and interconnection can be programmed by a configuration bitstream [13]. We used a Spartan II xc2s400 from Xilinx Corp. [15].

The results of the synthesis in terms of chip area (slices, flip flop slices and look up tables) and the speed execution

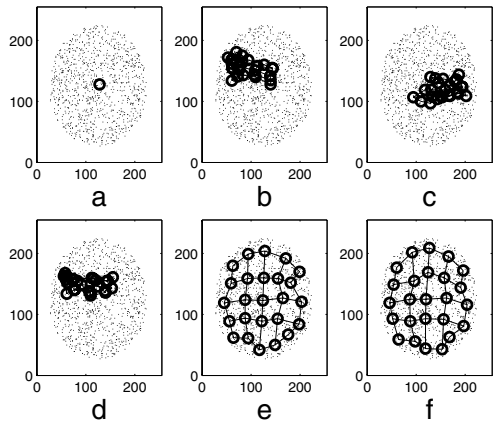

Figure 13. Input data for a simple clustering task (*small dots*) and weight vectors (*circles*) in the input space: (a) Iteration 0, (b) Iteration 1000, (c) Iteration 2000, (d) Iteration 3000, (e) Iteration 12000, and (f) Iteration 18000

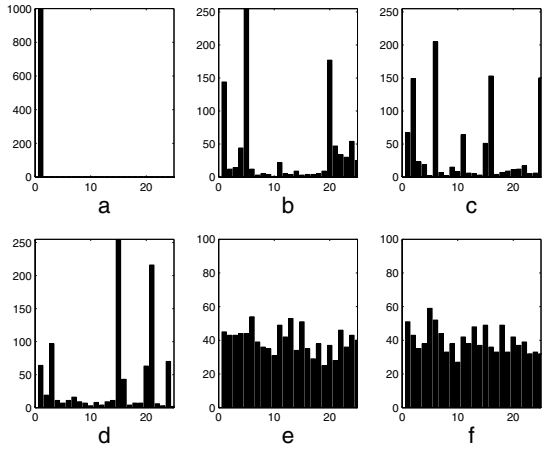

Figure 14. Histogram of winning events: (a) Iteration 0, (b) Iteration 1000, (c) Iteration 2000, (d) Iteration 3000, (e) Iteration 12000, (f) Iteration 18000

(maximum pin delay) are shown in Table 1. It must be said that no attempt was made to optimize the synthesis. Moreover, the xc2s400 is not even the largest device from the low cost FPGA family Spartan II. Virtex II family, for instance, offers up to 20 times more logic resources.

Table 1. Architectures Comparison

	One-dimensional Map	
	Cityblock	Chessboard
Slices	1105	1126
F.F. Slices	450	440
LUTs	1628	1691
Max. comb. path delay (ns)	35.24	35.99
	Two-dimensional Map	
	Cityblock	Chessboard
Slices	3009	2898
F.F. Slices	1095	1070
LUTs	4525	4450
Max. comb. path delay (ns)	35.04	36.01

Given that the maximum combinational path delay defines the commutation speed of the system and that the processing time per input vector is one clock cycle, the performance could be extracted from maximum combinatorial path delay for each system. The performance of the one-dimenional map with cityblock norm is 28.38 MCUPS, see table 1.

5.3 Discussion

Both Cityblock and Chessboard norms are suitable for hardware implementation because of the absence of multiplications. There was little change in the behavioral results due to a change in the metric. According to the synthesis results, however, the Cityblock norm seems to outperform the Chessboard norm in terms of both speed and area in the one-dimensional map. In the two-dimensional map, the Cityblock norm achieves a greater speed but consumes more area than the Chessboard norm-based architecture.

The proposed SOM architectures are compared with the SOM described in [4], Hikawa works over an Altera EP20K400EBC625 FPGA to implement the SOM and he describes his system by using VHDL. The Hikawa proposed SOM consists of a 5x5 neurons array with a 4.89 MCUPS of estimated performance and a maximum clock frequency of 200.2 MHz. Even though our architecture has less resolution (8-bits) compared with 10-bits for Hikawa SOM, the estimated performance is better than Hikawa SOM, this means that our architecture is a combinatorial design that use the clock synchronization just to update the weights. Likewise the advantage in area consumption of the Hikawa SOM, while our architecture use 360.000 equivalent gates to build the system, the Hikawa SOM use just 29.182 equivalent gates.

6 Conclusions and Future Work

Hardware-friendly architectures of both a one-dimensional and a two-dimensional Kohonen maps

were presented. Though very simple when compared with the original algorithm, the proposed models succeed in retaining the main features of the network, allowing data clustering with topology preservation. In particular, the change of metric, the restriction in the values of the learning rates and the definition of an exponential neighboring function were important to achieve the required simplicity.

The exponential neighboring function allows a greater flexibility than the step neighboring function, without incurring in the costs that the use of a gaussian neighboring function would involve. On the other hand, from a strictly behavioral point of view, the choice of whether a Cityblock or a Chessboard norm seems to be irrelevant. From the point of view of FPGA synthesis, however, the Cityblock metric seems to be preferable than the Chessboard metric, except if a two-dimensional, area sensitive design is taken into account.

Future work points to the use of the proposed architecture in real world problems, such image segmentation and compression and the construction of a perceptual map for mobile robot navigation. In order to cope with these problems, we must be able to process data of high dimensionality (only two-dimensional data were considered until now) and to implement large maps. We have already started working in the extension of the architecture so it can receive input data of any dimensionality (e.g, processing each component in a clock cycle) and in the definition of a local winner take all mechanism in order to allow a better scalability of the architecture. The idea is to be able to contruct large maps by the connection of several homogeneous chips implementing smaller maps.

References

[1] Asanovic, K.: A fast Kohonen net implementation for Spert-II. IWANN'97, Lanzarote, Canary Islands, Spain, June 1997.

[2] D. Ghosh , A. P. Shivaprasad: Possibilistic Clustering in Kohonen Networks for Vector Quantization Institute of Science, Bangalore, India.

[3] Hassoun, M. H.: Fundamental of artificial neural networks. MIT Press / Bradford Book (1995).

[4] Hiroomi Hikawa: FPGA implementation of self organizing map with digital phase locked loops. Neural Networks 18, 2005. pp. 514522.

[5] Kohonen, T.: Self-Organization and Associative Memory (3rd ed.). Springer-Verlag, Berlin (1989).

[6] Y. Liao: Neural Networks in Hardware: A Survey. Department of Computer Science, University of California, 2001.

[7] D. Macq et. al.: Analog Implementation of a Kohonen Map with On-Chip Learning. IEEE Transactions on Neural Networks, Vol 4, No 3, May 1993.

[8] R. Newcom, J. Lohn: Analog VLSI for Neural Networks. MIT Press / Bradford Book, 1995.

[9] Ruping S., Ruckert U., Goser K.: Hardware Design for Self Organizing Feature Maps with Binary Input Vectors. In: Lecture notes in Computer Science, Springer Verlag, 1993, pp. 488-493.

[10] T. Schnauer, A. et. al.: Digital Neurohardware. Technical University of Berlin, Berlin, 1998.

[11] Tamukoh, H., Aso, T., Horio, K. and Yamakawa, T.: Self-Organizing Map Hardware Accelerator System and its Application to Realtime Image Enlargement. In: Neural Networks, 2004. Proceedings. 2004 IEEE International Joint Conference on, pp. 2683-2687.

[12] M. Skrbek: Neural Networks-Hardware Implementation. Department of Computer Science and Engineerig, FEE CTU, Prague, 2000.

[13] Trimberger, S. M.: Field-Programmable Gate Array Technology. Kluwer Academic Publishers, 1994.

[14] Mentor Graphics. http://www.model.com

[15] Xilinx Corp. http://www.xilinx.com