A Mixed Hardware/Software SOFM Training System

Sistema Híbrido Hardware/Software para el Entrenamiento de Redes SOFM

Agustín Ramírez Agundis[1], Rafael Gadea Girones[2], Ricardo Colom Palero[2] and Javier Díaz Carmona[1]
[1]Department of Electronic Engineering, Instituto Tecnológico de Celaya, Av. Tecnologico s/n; 38010;
Celaya, Gto., Mexico; aagundis@itc.mx, jdiaz@itc.mx
[2]Department of Electronic Engineering, Universidad Politecnica de Valencia, Camino de Vera s/n, 46020;
Valencia, Spain; rgadea@eln.upv.es, rcolom@eln.upv.es

Article received on August 31, 2007; accepted on November 30, 2007

Abstract
This paper describes the design of a training system for a Self-Organizing Feature Map (SOFM). The system design aims two goals. The first is to reduce the training processing time by exploiting the inherent neural networks (NNs) parallelism through the SOFM hardware implementation. The second goal is to provide versatility to the training process by means of pre- and post processing of input and output data using Matlab-Simulink, which is also used as the software platform. The system uses as a coprocessor an FPGA based board connected via PCI bus at the host PC. To illustrate the system functionality we developed an application to analyze the effects over the map of scattering size in randomly generated weight initial values. When compared with the software approach for the same application, our system reduces the training time in 89%.
Keywords: Self Organizing Feature Map, Mixed Hardware/Software Implementation, Field Programmable Gate Array, Neural coprocessor.

Resumen
Este artículo describe un sistema para entrenar una red neuronal Self-Organizing Feature Map (SOFM). El diseño del sistema persigue dos objetivos. Primero, reducir el tiempo de procesamiento requerido para entrenar la red sacando provecho del paralelismo intrínseco de las redes neuronales mediante la implementación hardware de la SOFM. Segundo: proporcionar versatilidad al entrenamiento por medio del pre y post procesamiento de los datos de entrada usando Matlab-Simulink, también utilizado como plataforma del software. El sistema usa como coprocesador una tarjeta basada en un FPGA conectada a la PC anfitriona a través del bus PCI. Para ilustrar la funcionalidad del sistema se desarrolló una aplicación para analizar los efectos que sobre el mapeo tiene el tamaño de la dispersión de los valores iniciales de los pesos generados aleatoriamente. Cuando se compara con un sistema totalmente software para la misma aplicación, nuestro sistema reduce el tiempo de entrenamiento en 89%.
Palabras clave: Mapeo de rasgos auto-organizado, Implementación híbrida hardware/software, Arreglo de compuertas programables en campo, Coprocesador neuronal.

1 Introduction

Kohonen Self-Organizing Feature Maps (SOFM) [Kohonen, 1995] have been successfully used for a wide range of technical applications in fields like data analysis, pattern matching, and controlling tasks [Oja et al., 2003]. Neural Networks (NNs) require a huge amount of computations, mainly during the learning phase; therefore, software implementations of NNs algorithms have as a main drawback the impractical time-consuming sequential operation.

Exploiting the inherent parallelism of NNs, several kinds of devices have been used for designing fast NNs implementations, such as massively parallel computers, neuro-computers, and analog or digital full-custom or semi-custom ASIC's. A broad survey of parallel neural computing may be found in [Schoenauer et al., 1998].

Kohonen SOFM algorithm is particularly a highly parallel algorithm requiring much CPU time for network training with large data sets. In recent years, several SOFM parallel implementations have been developed [Hämäläinen, 2001] [Porrmann et al., 2006].

Kohonen SOFM training algorithm computes the neuron connections weights which depend on both the initial weight values, and the application order of training vectors. Therefore, Kohonen algorithm variants have been developed in order to reduce the influence of such issues, like the so-called Frequency Sensitive Learning [Xiao et al., 2002] that improves the mapping process.

Weight updating in Kohonen algorithm uses the learning rate and the neighborhood functions. Effective choices for these functions and their parameters have so far been determined only experimentally. This fact shows clearly the usefulness and convenience of having a platform that accelerate the process of SOFM networks training.

This paper expounds an enhanced SOFM training platform consisting of a mixed hardware/software system. The platform allows to maximize the resources utilization by combining a SOFM hardware implementation, exploiting its inherent parallelism, with software facilities, providing development flexibility with different training schemes and training parameters values.

The paper is organized as follows: the FPGA based hardware implementation of the SOFM is presented in section 2. In section 3 the system structure is described. The experimental work and the results are expounded in section 4. Finally, in section 5 conclusions and suggestions for future work are presented.

2 FPGA Based SOFM Design

Our SOFM [Ramirez et al., 2007]. consists of 16 units in the input layer and *N* neurons in the output layer. Fig. 1 illustrates the SOFM architecture. The design is divided into three sections: the processing units array, the address generator and the controller.

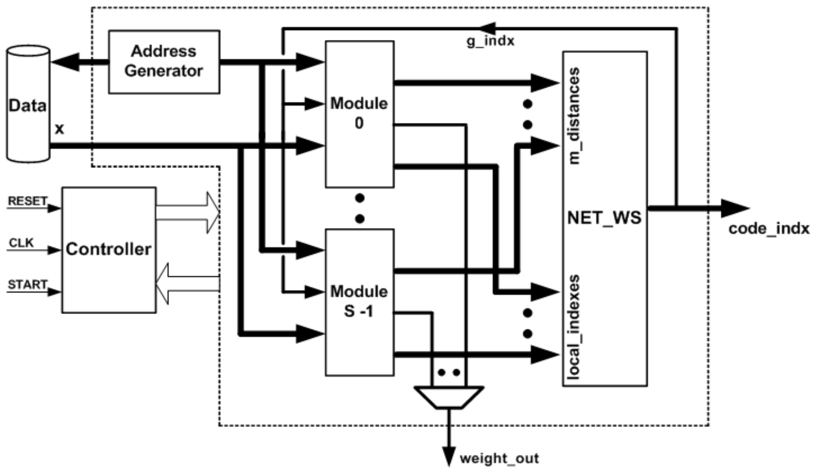

Fig. 1. Global hardware structure

The processing units array is distributed in modules, with 16 units each and a maximum of 16 modules. The amount of required modules *S* is selected at the synthesis stage by the parameter *indx_wide*=$\text{Log}_2 N$ bits. The block *NET_WS* compares the minimal distances obtained in the modules; its output is the winner neuron index (c*ode_indx* in Fig. 1). This index is split into two parts: (1) the *indx_wide*-4 most significant bits or global index, and (2) the four least significant bits or local index. The global index (*g_indx* in Fig. 1) determines which the winner neuron module is and the local index specifies which neuron is the winner in that module. To be used at the update stage, the global index is back-connected to all modules.

2.1 Address Generator

To process an input vector, its elements must be read from an external memory and applied as the network input, one element at a time, until the complete vector has been scanned. Each input vector element is processed along with its respective weight vector element in all the net neurons simultaneously. At the training stage, the scanning is done twice, first to determine the winner neuron and second to update the weights for the winner neuron and for all the neurons that are located in its neighborhood region. The address generator block has counters addressing the input vectors set, the input vector elements and the weight vector elements.

2.2 Processing Units Array

Our hardware design is based on node parallelism; thus, the circuit includes a processing unit for each neuron network. Fig. 2 shows the module structure. The module has three inputs: *x* is the input vector element being processed, *addr* is the address for the synaptic weights RAM location where the corresponding weight element is stored, and *G_index* indicates which module contains the winner neuron. The module outputs are the winner neuron local index *loc_indx* and the distance between the input vector and the weight vector associated with the winner neuron, *m_dmin*. Each module consists of 16 processing units, *UNIT_0* to *UNIT_15*, and the *M_WS* comparator that identifies the module winner neuron.

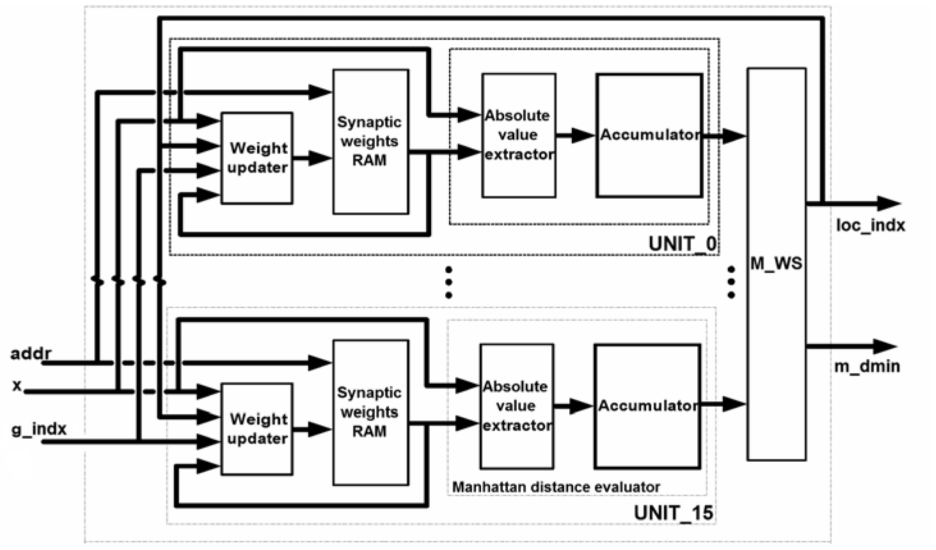

Fig. 2. Structure of a module with 16 units

2.2.1 Synaptic Weights RAM

Each weight is stored as an *I*-bits integer plus an *F*-bits fractional part and given that the net has 16 input neurons, for every output neuron it is needed a block of RAM with 16 locations, *I*+*F* bits wide.

NNs learning can be done by adaptive training or batch training. Original SOFM algorithm consists only of adaptive training in which the weights updating are done immediately after each training vector is presented. Kohonen later proposed the so called batch map algorithm [Kohonen, 1993], in which the weights of the network are updated until the entire training set has been applied to the network. The SOFM in our design can work with both kinds of training. In batch training, the synaptic weights for each neuron use two blocks of RAM. At even epochs (an epoch is the process required to apply the entire training set to the network), one of the blocks is used to find the winner neuron and updating of weights is done over the other block. At odd epochs, the use of RAM blocks is reversed. For adaptive training, only one RAM block is needed.

2.2.2 Weight Updating

According to the Kohonen SOM training algorithm, if *g* is the winner neuron then the weight of any neuron *u* is updated using Eq. (1).

$$w_u(t+1) = w_u(t) + \alpha(t)\eta(t, d_{u,g})(x(t) - w_u(t)) \quad (1)$$

In Eq. (1), $\alpha(t)$ is the learning rate, and $\eta(t,d_{u,g})$ is the neighborhood function. The values of $\alpha(t)$ and $\eta(t,d_{u,g})$ decrease with time. Additionally, the neighborhood function value becomes smaller as long as neuron *u* is located farther from neuron *g*. The term $d_{u,g}$ is the Manhattan distance between neurons *u* and *g*.

The SOFM topology uses a two-dimensional pattern with rectangular grid and Manhattan distance as it is shown at Fig. 3. We use rectangular neighborhood function due to it is the most appropriate for hardware implementation.

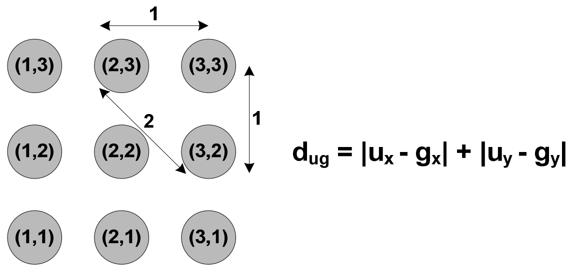

Fig. 3. SOFM topology

In this design, the learning rate and the neighborhood function values can only be negative powers of two, in such a way multiplications become shift operations. Thus, those functions can be described by Eq. (2).

$$\alpha(t) = \begin{cases} 2^{-(k+\beta)} & if \quad 1 \le k + \beta \le 7 \\ 2^{-7} & if \quad k > 7, \end{cases} \qquad \eta(t, d_{u,g}) = \begin{cases} 1 & if \quad d_{u,g} = 0 \\ 0 & if \quad d_{u,g} > 1 \\ 2^{-1} & if \quad 1 \le k \le 6, d_{u,g} = 1 \\ 0 & if \quad k > 6, d_{u,g} = 1, \end{cases} \tag{2}$$

where *k=round(t/4)*, being *t* the epoch number. Constant β fixes the initial value of $\alpha(t)$. The value of β is an input to the hardware and is set by the software interface as part of the control word (Fig. 6).

The hardware implementation described by Eq. (1) and Eq. (2) is very simple. Values of $x(t)-w_u(t)$ right-shifted up to seven times are applied to the inputs of a multiplexer. The multiplexer output is selected based on the number of elapsed epochs and the neighborhood function value. Finally, this output is added to $w_u(t)$. Both, the subtractor and the adder are *I*+*F* bits wide.

Our SOFM design uses Frequency Sensitive Learning method to improve the training process. There is a counter for every neuron. The counter is incremented each time the corresponding neuron becomes the winner. The counter content is five bits right shifted and the result is added to the distance between the corresponding neuron and the input vector. Therefore, neurons that are frequently winner are penalized.

2.2.3 Manhattan Distance Evaluator.

Manhattan distance was selected for two-vector distance measurement. It is also used to compute the two neurons distance needed for the neighborhood function. Thus, the distance between the two *k*-dimensional vectors *X* and *W* is evaluated using Eq. (3).

$$d_{X,W} = \sum_{i=1}^{k} |X_i - W_i| \tag{3}$$

In this design, *k*=16 and the absolute values are *I* bits wide; therefore the accumulator is *I*+4 bits wide.

2.2.4 Distance Comparator

The *M_WS* block is a four-level tree of multiplexers/comparators. Each of them receives two distances, one associated to an even-index and the other associated to an odd-index. At its output, every multiplexer/comparator provides the minimum of these two distances and a bit-index whose value is zero if the even-index distance is lower than or equal to the odd-index distance; otherwise, this bit-index is equal to one. The minimum distance and its associated bit-index are propagated from one level to the next in such a way that the entire index and the minimum distance appear at the fourth level. The net winner selector block, *NET_WS* in Fig. 1, has the same architecture; the only difference is that the number of tree levels depends on the output neurons quantity and will be between one and four.

2.3 SOFM Controller

The SOFM controller is based on a finite-state machine. Fig. 4 shows the state transition diagram for the controller. Actually, each state represents a net operating stage controlled by a finite-state sub-machine. The first stage is intended to fill the weight vectors with initial values. After the weight vectors have been filled, the net proceeds with the training stage, and this is executed iteratively until the predetermined epoch quantity is reached. Finally, the net provides at its output the obtained final values of weight vectors, which are ready to be saved in an external memory device.

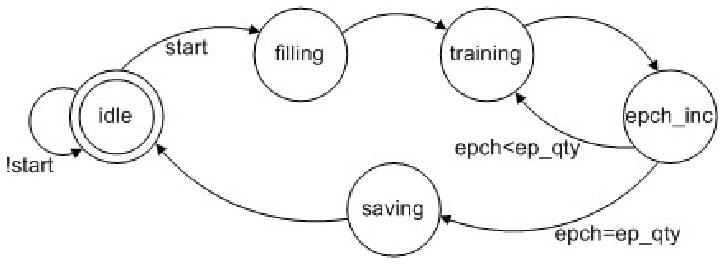

Fig. 4. SOFM controller operating stages

An input vector training consists of two steps: neuron-winner searching and weights updating. The neuron-winner searching step has two time intervals; first, the absolute values of the differences are extracted and accumulated; and second, the smallest result is selected and the corresponding index value is generated. The neuron-winner searching step takes 18 + *indx_wide* clock cycles: 18 for the first time interval and *indx_wide* for the second. The weights updating step takes 19 clock cycles. Hence, an entire input vector training requires a total of 37 + *indx_wide* clock cycles.

3 System Architecture

The SOFM design was developed and implemented on the XC2V3000 Virtex-II FPGA located in the ALPHA-DATA ADM-XRC-II board [Alpha Data, 2005], which is used as a coprocessor in the host computer. A SOFM with 16 input neurons, 128 output neurons, 10 bits input data and synaptic weights 18 bits wide could be implemented in this FPGA device. Fig. 5 sketches an outline of our system. The system uses two RAM memory banks and the status and control registers contained in the board. The board communication with the host computer is done through a PCI bridge.

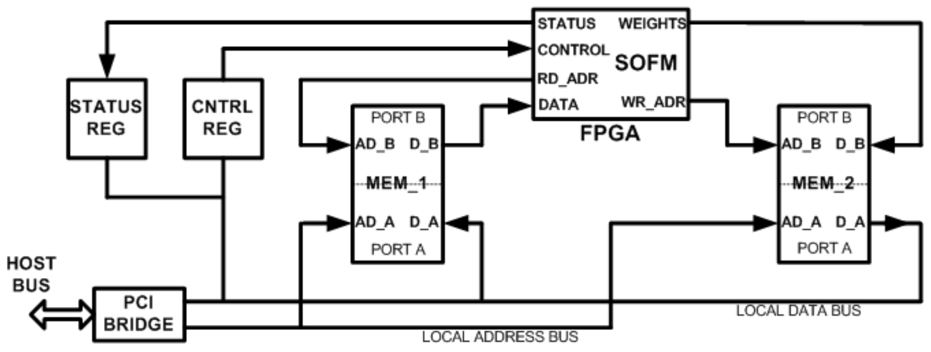

Fig. 5. System architecture outline

Each memory bank provides 1M x 32 bits true dual-port static synchronous RAM with a reading latency of 3 clock cycles. Data transfers between the host and memory banks are done through the PCI bridge using the local address and local data busses, which are connected to port A pins. The FPGA device is directly connected to memory banks using port B pins.

Before running a training task, the training vector set and the weight vector initial values are downloaded to memory bank *MEM_1* by the host computer, which also provides the training parameter values and the start signal by means of the control register. Fig. 6 shows the template for the control word.

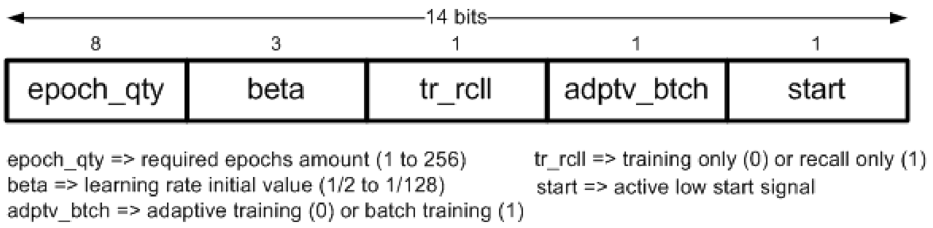
14 bits
8
3
1
1
1
epoch_qty
beta
tr_rcll
adptv_btch
start
epoch_qty => required epochs amount (1 to 256)
beta => learning rate initial value (1/2 to 1/128)
adptv_btch => adaptive training (0) or batch training (1)
tr_rcll => training only (0) or recall only (1)
start => active low start signal

Fig. 6. Control word template

At the end of the training task, the final weight vectors and the winner neuron index for each input vector obtained at the last epoch are written by the FPGA into memory bank *MEM_2*, and finally an end of task signal is issued to the host by means of the status register. The host then reads out both the final weight values and indexes from memory bank *MEM_2*, and proceeds with the post-processing. If it is necessary, the host can start a new task using different initial values, another training set or different learning parameter values.

Taking into account the memory banks reading latency, 148 + *indx_wide* clock cycles are required to train the SOFM net with one input pattern. The clock signal driving the FPGA in the board was set to 40 MHz, and the PCI bridge clock was set to 33 MHz. Our SOFM performance, both speed and resource occupation, can be found at [Ramirez et al., 2007].

4 An Application: Scattering of Initial Weight Values

When training SOFM nets, final performance can significantly depend on the weights initialization. Ideally, a certain amount of SOFMs with varying random seed needs to be created, their resulting maps analyzed, and the map with the training vectors best-balanced distribution should be chosen. This is a lengthy and laborious task, so far not automated. For large data sets it becomes impractical, therefore in practice only one map is generated and it is accepted as the final result.

The proposed system was used to experimentally analyze how the scattering of randomly generated map initialization values affects the training of a SOFM net. The system was configured to execute a series of 10 training sessions with the same net. For each training session a different map initialization was used. The initial values were generated randomly within a centered window in 128, half of the used range for the input data value, and with a growing width from one session to the next. For the first training session the width of the window equals 9 (the initial weight values are between 124 and 132), for the second session the window width equals 17, and so on until reaching a width of 81 for the tenth training session.

A SOFM with 64 output neurons and 16 input neurons was used. The 256 x 256 pixels Lena image was partitioned into 4096 image blocks of size 4x4 and the generated vectors were used as the training set. The adaptive kind of training was used. Each session consisted of 120 epochs (491520 iterations).

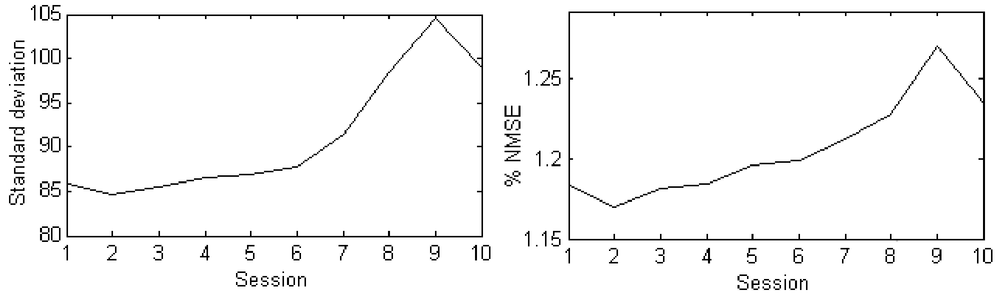

Fig. 7. Training sessions comparison

Fig. 7 shows the results. For each session, the obtained indexes were used to count the number of times every output neuron was the winner at the last epoch, then its standard deviation was obtained, all these operations were done by Matlab functions. The graph on the left is the plot of the standard deviation values versus the training session number. For each session, we also obtained the differences between every training vector and the weight vector of the corresponding winner neuron, and we calculated the normalized mean square error (NMSE). The graph on the right shows the NMSE percent values.

Results in both graphs are consistent. The best training occurred at session two; therefore the weight initialization is better when the randomly generated initial values are in the range between 120 and 136. This scattering of the initial weight values yields to the best training process because the so generated map has the most uniform distribution and simultaneously minimizes the NMSE value of the distortion.

Using our hardware/software mixed system running at 40 MHz, the time required to complete the ten training sessions was 62 seconds. When the same series was simulated using the SOM Toolbox [Vesanto et al., 1999], running over a 2.8-GHz Pentium D system, it took up 572 seconds, which is 9.2 times longer. Therefore, our hardware/software mixed system reduces the training time in 89% in comparison with the entirely software system.

5 Conclusions and Future Work

A hardware/software system for training SOFM NNs was presented. The FPGA based hardware implementation of the SOFM reduces the time required for its training. Since the used FPGA is located into a PCI board, it becomes a neural coprocessor for the host computer. By using Matlab-Simulink as the software platform, it is possible incorporate directly existent pre and post data processing functions written in Matlab or C. An application was developed to analyze the effects over the map of scattering size in randomly generated weight initial values. When compared with the SOM Toolbox software for the same application, our system reduces the training time in 89%.

Future work will be focused on extending the software tasks beyond the data pre and post processing. Thus, software functions will be incorporated to analyze and adjust the training in the course of its development. With this purpose, we will incorporate into the system an embedded microprocessor, which will be located closer to the neural coprocessor than the host processor.

References

1. **Alpha Data Parallel Systems LTD.**, "ADM-XRC User Manual", available at www.alpha-data.com; 2005.
2. **Hämäläinen, T. D.**; "Parallel Implementations of Self-Organizing Maps", in Seiffert, U., Jain, L. C., eds., Self-Organizing Neural Networks: Recent advances and applications, Vol. 78, pp. 245-278; Springer-Verlag, 2001.
3. **Kohonen, T.**; "Things You Haven't Heard about the Self-organizing Map", IEEE International Conference on Neural Networks, 1993, Vol. 3, pp. 1147-1156, 1993.
4. **Kohonen, T**.; Self-Organizing Maps; Springer-Verlag; Berlin; 1995.
5. **Oja, E., Kaski, S., Kohonen, T.**; "Bibliography of self-organizing map papers: 1998-2001 Addendum"; Neural Computing Surveys, Vol. 3, pp. 1-156; 2003.
6. **Porrmann, M., Witkowski, U., Rückert, U.**; "Implementation of Self-Organizing Feature Maps in Reconfigurable Hardware", in Omondi A. R., Rajpakse J. C., eds., FPGA Implementations of Neural Networks, pp. 247-269, Springer, 2006.
7. **Ramirez, A., Gadea, R., Colom, R.**; "A hardware design of a massive-parallel, modular NN-based vector quantizer for real-time video coding"; Accepted at Elsevier Journal of Microprocessors and Microsystems; doi:10.1016/j.micpro.2007.06.004, Available online 5 July 2007.
8. **Schoenauer, T., Jahnke, A., Roth, U., Klar, H.**; "Digital Neurohardware: Principles and Perspectives"; Proceedings of Neuronal Networks in Applications '98, pp. 101-106; Magdeburg 1998.
9. **Vesanto, J., Himberg, J., Alhoniemi, E., Parhankangas, J.**; "Self-organizing map in Matlab: the SOM Toolbox"; Proceedings of the Matlab DSP Conference 1999; pp. 35-40; Espoo, Finland, Nov. 1999.
10. **Xiao, R., Chang, C. H., Srikanthan, T.**; "An efficient learning rate updating scheme for the self-organizing feature maps"; Proceedings of 2nd Int. Conf. on Visualization, Imaging and Image Processing; pp. 261-264; Malaga, Spain, Sep. 2002.

Agustín Ramírez Agundis *received the M. Sc. degree from the Universidad de Guanajuato, Mexico, in 1977. Since 1986 he has been a lecturer at the Department of Electronics Engineering at the Instituto Tecnologico de Celaya, Mexico. Currently he is pursuing the Ph.D. degree at the Universidad Politecnica de Valencia, Spain. He is working in Hardware Design of Neural Networks for Image Compression.*

Rafael Gadea Girones *received the M.Sc. and Ph.D. degrees from the Universidad Politecnica de Valencia, Spain, in 1990 and 2000, respectively. Since 1992 he has been a lecturer of the Department of Electronics at the Universidad Politecnica de Valencia. Currently, he is assistant professor at Telecommunications Engineering School of the Universidad Politecnica de Valencia, Spain. His areas of research interest include hardware description languages, design of FPGA-based systems and design of neural networks and cellular automatas.*

Ricardo José Colom Palero *received the M.Sc. and Ph.D. degrees from the Universidad Politecnica de Valencia, Spain, in 1993 and 2001, respectively. Since 1993 he has been a lecturer of the Department of Electronics at the Universidad Politecnica de Valencia. Currently, he is assistant professor at Telecommunications Engineering School of the Universidad Politecnica de Valencia, Spain. His areas of research interest include VLSI signal processing, design of FPGA-based systems and custom digital signal processing for audio and video applications.*

Javier Díaz Carmona *received the M. Sc. degree in 1997 and the Doctor in Science degree in Electronics in 2003 from the National Institute of Astrophysics Optics and Electronics (INAOE), Puebla, Mexico. Since 1991 he is part of the faculty of Electronics Engineering Department at the Instituto Tecnologico de Celaya, Mexico. His research interests include Digital Signal Processing, filter design techniques and Microcontroller, DSP/FPGA embedded systems applications. He is member of IEEE Signal Processing Society and National Researcher System (SNI) Mexico.*