Reconfigurable Hardware Architecture of a Shape Recognition System Based on Specialized Tiny Neural Networks With Online Training

Félix Moreno, Jaime Alarcón, Rubén Salvador, and Teresa Riesgo

***Abstract*—Neural networks are widely used in pattern recognition, security applications, and robot control. We propose a hardware architecture system using tiny neural networks (TNNs) specialized in image recognition. The generic TNN architecture allows for expandability by means of mapping several basic units (layers) and dynamic reconfiguration, depending on the application specific demands. One of the most important features of TNNs is their learning ability. Weight modification and architecture reconfiguration can be carried out at run-time. Our system performs objects identification by the interpretation of characteristics elements of their shapes. This is achieved by interconnecting several specialized TNNs. The results of several tests in different conditions are reported in this paper. The system accurately detects a test shape in most of the experiments performed. This paper also contains a detailed description of the system architecture and the processing steps. In order to validate the research, the system has been implemented and configured as a perceptron network with back-propagation learning, choosing as reference application the recognition of shapes. Simulation results show that this architecture has significant performance benefits.**

***Index Terms*—Neural network hardware implementation, run-time learning, recognition.**

I. Introduction

ONE OF THE major problems in computer vision is to build systems with the ability to identify shapes in real world scenarios [1]–[7]. The target application of this paper is the correct identification of road traffic signs in images taken by a car-mounted camera [8], [9]. The basic technique used for this in most applications, known as pattern matching, is to compare each portion of an image with a set of known models. The approach taken in this paper is to use specialized tiny neural networks (TNNs), which are explained in Section III, making it possible to use a massively parallel architecture efficiently. One of the most important features of artificial neural networks (ANNs) is their learning ability. Size and real-time considerations show that on-chip learning is necessary for a large range of applications [3].

Neural-network-based recognition systems have several levels of inherent parallelism. Traditional software implementations cannot implement these parallel working schemes, unless done on multiprocessor systems. Since the system proposed in this paper is implemented in hardware, this inherent parallelism can be exploited. The general architecture of the system is shown in Fig. 1.

This higher level of parallelism achieved with the hardware implementation, as opposed to most software attempts, allows making several computations concurrently. Therefore, a higher processing throughput is obtained [10]–[13]. In addition, the hardware implementation is also highly portable [14], due to the generic hardware description employed.

ANN implementations can be classified in two main categories: software running on a microprocessor, digital signal processors, or general-purpose processors, and hardware architectures based on application specific integrated circuits or field-programmable gate arrays (FPGAs) [15].

The microprocessor-based implementation is more flexible and relatively easy to implement. However, when the network becomes larger, it is not the best option as regards the processing time.

The number of "synapses" and multipliers needed in a fully interconnected network is proportional to the squared total number of neurons. The speed slows down due to the increase in the number of multipliers, and the chip area required increases significantly, which becomes one of the critical problems in ANN design. In order to solve this chip size issue, the use of hardware multipliers seems to be an option. In addition, neural networks with reusable multipliers, or even without them at all, may be designed [16], [17].

This paper explores multiplier reusability based on an internal bus structure. Taking into account the parallelism of the neural network model, it is possible to map the architecture on array processors, obtaining a linear growth in the number of multipliers. Therefore, we are before a suitable scenario for ANN hardware implementation in embedded systems. Fig. 2 shows a network interconnected by means of an array processor model [18]–[21], where w_i and x_i are the inputs to each processor. In the case of a neural network, these are the weights and data inputs of each neuron, respectively.

The main objective of this paper is the design of a reconfigurable efficient low-cost architecture for shape recognition.

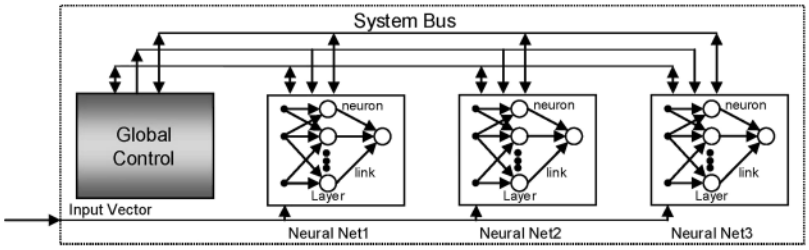

Fig. 1. General architecture of the system.

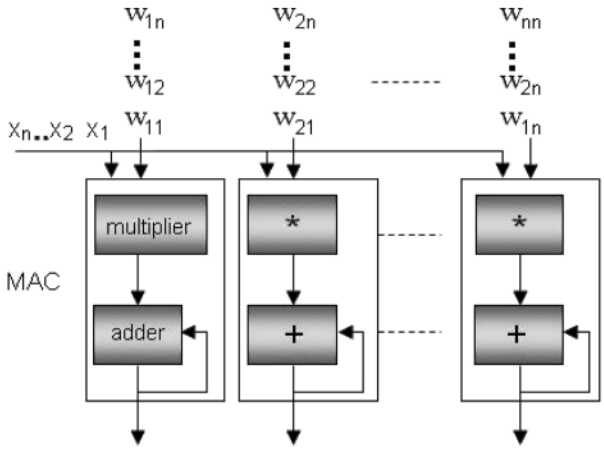

Fig. 2. Array-processor-based architecture.

Robust methods for the analysis of images, and the implementation of a system based on specialized TNN have been developed for shape recognition by means of the analysis of characteristics elements of shapes, called singularities [22], [23], and which will be explained later in this paper, in Section V. Traffic signal and/or pedestrian recognition are two of the most relevant applications. These networks work cooperatively, as explained later in this paper, to obtain the classification of the image.

The main restriction comes with the complexity of the information contained in the image data, very sensitive to changes in the environment. Therefore, it is necessary to develop a recognition system that allows for dynamic reconfiguration [1], [24].

The scarce resources provided by the hardware, imposed by the low-cost design basis, require an architecture developed for optimal use.

The system uses a low-cost CMOS camera. The images are processed by an Altera Cyclone II FPGA that does median and Sobel filtering of the incoming frames at PAL rate [720× 576 size images at 25 fps (frames per second)]. The result of this preprocessing stage is the input to the TNNs, also implemented in the Altera FPGA and built up by small perceptron multilevel networks.

To comply with TNN, and in general, neural network ability to learn, a side-by-side online learning system has been built in the embedded PowerPC processor in a Xilinx Virtex II device. To allow a faster adaptation of the system, an initial training was done with a simulation model implemented in Matlab that helped when validating the functional behavior of the system. After the PowerPC learning algorithm implementation was finished, another training phase was done, comparing it with the results obtained in Matlab. These results were then used to configure the networks in the FPGA.

The whole architecture of the system is shown in Fig. 3.

II. FPGA Implementation of the General System Architecture

The requirements of recurrent learning processes can be satisfied by the flexibility offered by FPGAs reconfiguration capability, [11], [25], [26]. Weight modification and architecture reconfiguration can be carried out at run-time.

When analyzing ANN hardware implementation, the following considerations should be taken into account: frequency, precision, configuration issues, and parallelism degree to implement. In order to improve general system features, two units have been designed: basic and control units.

Basic units (specialized neural networks) are in charge of signal processing and weight and bias data storage. This includes all the required operations for neural computations, as imposed by the neuron model implemented, shown in Fig. 6: weights by inputs products, accumulation, and nonlinear function activation. The control units are in charge of signal transmission, making the parallel processing of the algorithm possible.

The proposed architecture has been efficiently mapped to hardware from its algorithmic functional high abstraction level description, making it suitable to be implemented on an FPGA device. This stated efficiency is shown in Section V.

A. Learning Algorithm Segmentation

To accomplish the learning operation the algorithm is divided in three phases, known as: feedforward, back-propagation, and update [27]–[30]. In the feedforward phase, the input signals propagate through the network from one layer to another, eventually producing some response in the output of the network. This response is compared with the desired (target) response, generating error signals that are propagated in backward direction through the network. In this backward phase of operation, the free parameters of the network are adjusted to minimize the sum of square error. Finally, weights and biases are updated using the data obtained in the previous phase. The process is repeated as many times as necessary in order to have a trained network. Usually, this process is made using general-purpose computers, and is known as offline training. The three phases of the algorithm are shown in Fig. 4.

Since the proposed architecture is self-reconfigurable at run-time, independent and separated modules for each of the stages where developed [11], [12], [31].

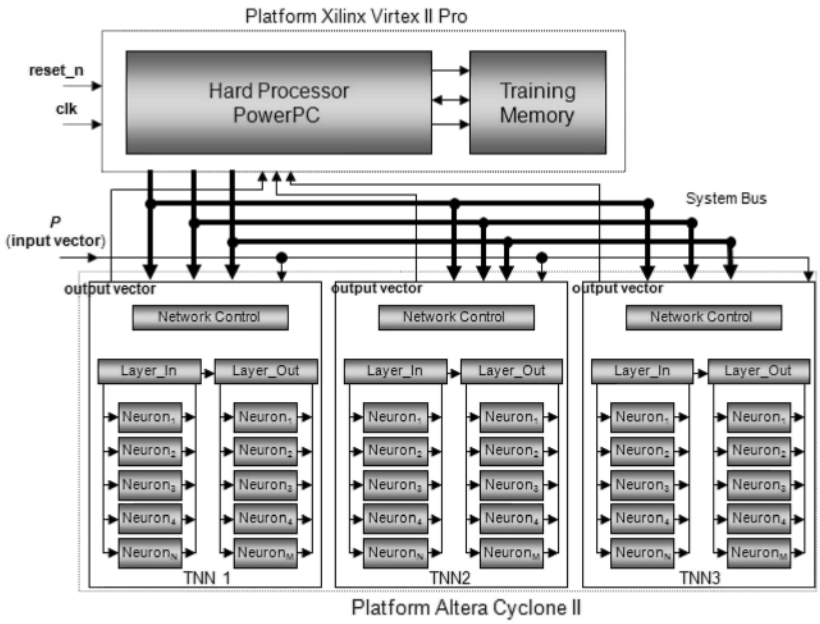

Fig. 3. System architecture.

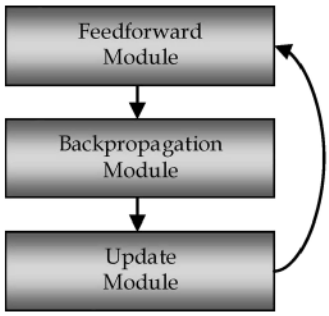

Fig. 4. Sequential algorithm for learning operation.

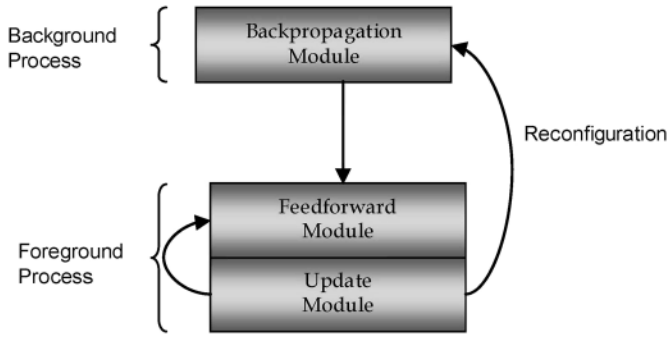

Fig. 5. Algorithms segmentation for learning operation.

It must be stated here that when using the word "reconfiguration" within this paper, it is not referred to an actual FPGA fabric reconfiguration, but a functional reconfiguration based on some parameter update. Therefore, neither the network structure is changed at all nor the number of neurons modified.

For the system to carry out an online adaptation [3], [19], the same learning rules should always be applied concurrently over a new pattern. When the network is reconfigured, the control unit executes the learning process concurrently, using the training patterns stored in the previous learning phase along with the new pattern to be recognized. This pattern is the one that triggered this learning phase.

When a learning process finishes, collected data are transmitted to the weights and bias network memories. This is carried out by the control unit, which makes them flow through the back-propagation level, directing this learning phase and making the FPGA get adapted.

Fig. 5 shows the implementation of the different levels of the learning algorithm. The feedforward and update modules corresponding to the basic unit were implemented in the same FPGA (Altera Cyclone II), being executed concurrently as a foreground process.

To accomplish this online learning, when the network finishes the feedforward operation, a dedicated module (to be explained later) computes the recognition uncertainty. If this computation exceeds an empirically set threshold, a request is sent to the control unit that triggers a learning phase of that network. The input responsible of this network response is therefore considered as a new pattern. The back-propagation module, actually responsible for the online learning stage, was implemented as a background process in the embedded PowerPC processor in a Xilinx Virtex II device (virtex2p xc2vp30). Each module of the algorithm follows the general architecture of the system proposed and showed in Fig. 3.

By means of a state machine, three modes of operation of the system were defined. In the *initialization mode*, the system loads the initial values of the weights and biases from memory, entering afterward into the *classification mode*. In this state, the network works in feedforward. When, as explained above, it detects a new pattern to apply to a new learning phase, it changes to the *reconfiguration mode*. When this is over, the update is carried out in order to enter again into a new classification stage. The different modes of operation and the state machine will be explained later in this paper.

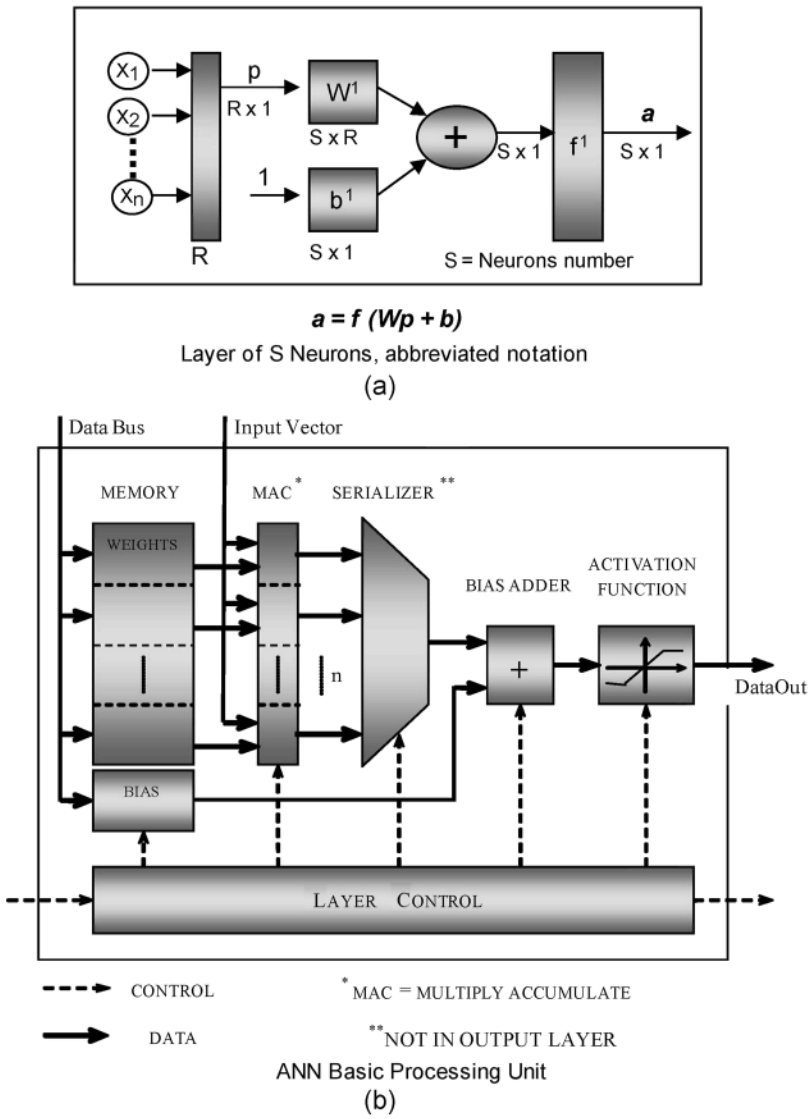

Fig. 6. TNN model. (a) Mathematical model. (b) Hardware model.

III. Specialized TNNs

Taking into account the problems of size and scalability, we propose a design based on the mathematical model [28] of the neural networks, similar to one shown in Fig. 6(a).

As explained above, the synapse number is limited (network size) by the FPGA resources available (mainly memory and area) [32]. In addition, the network architecture (number of neurons and number of layers) is also limited by these hardware resources [33]. In order to overcome these difficulties, a basic processing unit (BPU) is suggested as the central component of the network, which can be modified to feature one or several neurons. Each layer of the network is composed by several of these BPUs, obtaining different topologies according to the configuration of the internal registers of the system.

Fig. 6(b) shows the model of a BPU. The hardware architecture is obtained by directly mapping the high-level functional model of the perceptron neural network into its equivalent hardware representation. The data input vector comes from the preprocessing stage, while the weights data bus comes from the training memory as shown of Fig. 3. The control of the layers has been implemented so that each layer is self-controlled but directed by the previous one, giving the system a general control strategy implemented in a distributed fashion [8]. This makes it possible for each module to have control over itself, but making the overall system work coordinately among the different modules, allowing the system to show the emergency of an incipient intelligent behavior (distributed intelligence).

Since a low-cost system is to be designed, as mentioned in the introduction, a device with scarce (limited) hardware resources has been chosen. Therefore, a tradeoff between performance and hardware resources consumption has to be met. With the suggested model for the BPU, a nearly fully parallel architecture has been achieved.

All hardware neurons are formed by a MAC unit (multiplier and accumulator), a serial unit (multiplexer), and the nonlinear activation function modules, all of them interconnected by a parallel system bus, as shown in Fig. 6(b).

MAC units are connected through the internal data bus to their weights memories and to the input data serial stream (input vector). Supposing an N-neuron input layer, this architecture

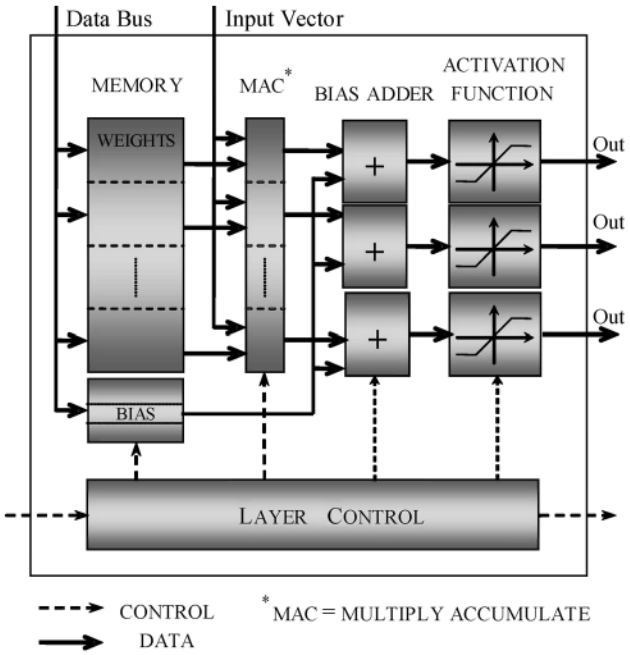

Fig. 7. Output layer of a basic unit.

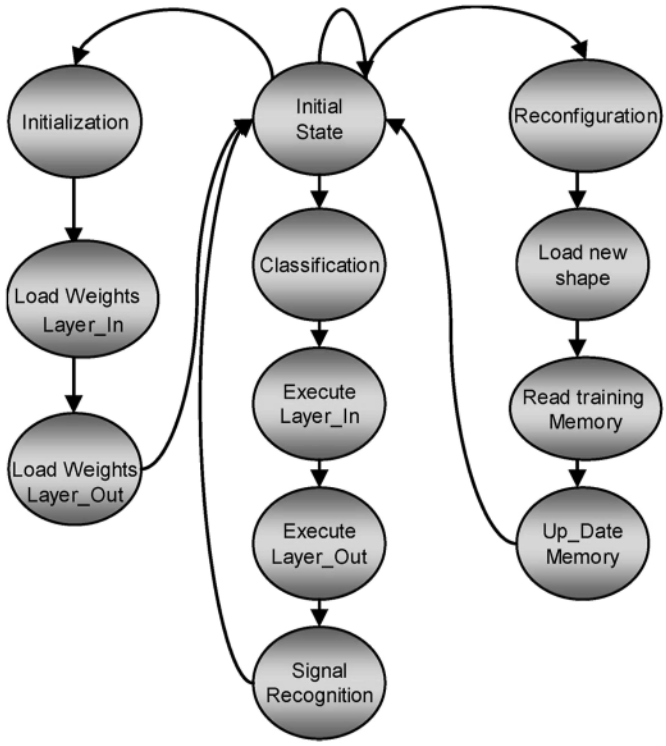

Fig. 8. State machine defining system operation.

allows carrying out N operations in parallel with each input datum, due to the simultaneous access to the memories through the internal bus structure. Therefore, the weights and bias memories have been implemented in the RAM modules embedded in the FPGA (Altera M4K blocks). These modules allow being accessed independently since they can be split in blocks (never smaller than 4096 b through the various memory width-depth combinations possible), so faster (the fastest possible in terms of concurrent accesses to the memories) memory accesses are achieved thanks to this distributed memory scheme.

As mentioned, a tradeoff has to be observed between cost and performance. Neural networks have a parallel connection between different layers. This would imply a huge resource consuming architecture, so some data serialization is needed. To further optimize the architecture, due to this *mandatory* serialization, just one adder and one activation function have been implemented. This way, a BPU has an output data vector, used as the input vector of the following BPU, i.e., layer.

A reduction of $N-1$ bias adders and $N-1$ activation functions is achieved. From the point of view of the information transfer through the different network layers, the BPU architecture can be seen as a black box with an input and output vector (feedforward) structure. Due to this resource-sharing scheme, it is possible to implement several multilayer perceptron-type neural networks [21].

As a special case, when talking about a perceptron multilayer network with few neurons in the output layer, it is not necessary to do any serialization at all. Therefore, in output BPUs, there is an adder and an activation function per neuron. The bias memory has been implemented as registers within the adder, what avoids wasting most of the memory bits of a RAM block. The architecture of an output layer unit is shown in Fig. 7.

With the described architecture for the BPUs, it is easy to build up a neural network by simply interconnecting two or more of these units, depending on the number of layers required. The interlayer connection is performed by sticking together data and control buses from each layer. The data flow is controlled by the control unit explained above. This hardware architecture of the network layers confers the system modularity and scalability features. This may be helpful for future and more powerful versions, implemented on bigger FPGAs.

The state machine of the Control Unit has been carefully designed to improve the system performance. Fig. 8 shows the three different branches that implement the functional behavior previously described.

The design of the system architecture allows for several networks to work (classify) in parallel. Moreover, the training process could be executed in another TNN concurrently.

In order to maintain the processing speed of a TNN in hardware and its versatility in simulations, the reconfiguration of the neural network on its different hardware levels has to be possible. Different researches have revealed that general-purpose processors can be used in order to reprogram the neural network. We could also use FPGAs to modify the bus structure and the BPU by means of a change in the configuration registers [26].

Focusing on the two main ANN hardware implementation possibilities, the features of general-purpose processors make them more adequate in terms of *programming* easiness, although they have two important drawbacks: the slow processing speed and the required area. On the other way, reconfigurable hardware networks are harder to *configure* but achieve higher processing speeds, due to its parallel architecture, while using a smaller chip area. Moreover, they can be included in an integrated circuit as a system-on-chip. Due to the characteristics of the proposed system, it can be considered to be a heterogeneous

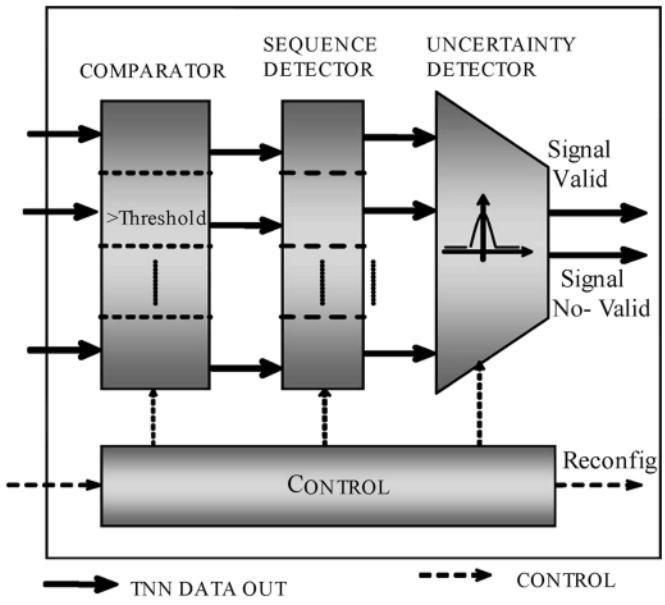

Fig. 9. Uncertainty module.

architecture, combining the best of both approaches: the programming easiness of general-purpose processors and the intrinsic parallel architecture of FPGAs.

The specialized network design has an *Uncertainty* computation stage. Its main function is checking that the output data of the networks are somehow related to the output obtained with the training patterns, validating the recognition process. This happens when the uncertainty module reports a probability higher than 75%. On the other hand, if it is in a range between 50% and 75%, a reconfiguration request is asserted [34]. These probability levels have been empirically set. This way, the system is aware of what network was supposed to have identified the object, and therefore, train that network again with the new pattern (input data just analyzed). The stages of this uncertainty module are shown in Fig. 9.

When this request is asserted, input data are acquired and attached to the appropriate training memory (initially filled with the patterns of the initial online training) as a new pattern for the following training process. Since the networks are trained to identify singularities, the data stored in memory are the pixels related to these singularities, not the entire image.

When this online training (actually, this training is an adaptation of the network, because it was initially trained, but is now retrained to get adapted, taking advantage of the initial training) is finished, the hardware modules have been reconfigured: training memories (content and dimensions), as well as weight and bias memories have been updated, with new values obtained at the end of the process.

IV. System Architecture—TNN and Control Integration

The designed system is highly parallel, so it is able to execute several tasks at the same time. The networks in our system are also cooperative, so they are able to solve complex issues through the contribution of each small network. As an example of the system application, the networks can be trained to identify characteristic elements of shape (singularities) such as right-angled corners, round segments, and acute-angled corners. These singularities are used for the recognition of rectangular, circular, and triangular shapes. Autonomous robots or intelligent systems for cars may use this kind of system [23].

The general architecture of the system, and its control strategy, as shown in Fig. 1, has been conceived so that it can easily be adapted to different applications. Due to this design requirement, the core of the architecture is based on an efficient and robust shape recognition system that allows its adaptation to different recognition tasks with little tuning in the system architecture.

The decision to have the communication of the control system through a bus structure was taken after consideration of the efficiency level that we wanted to achieve. In this way, the memory blocks share the same space on the system and can be accessed with a logic address, obtaining as a result a distributed memory system on the networks with a centralized control. The addressing mode was considered to be the optimum model because it does not require a redundant memory for the networks, and only during the reconfiguration process may exist a redundancy in the network memories that have to be reconfigured, achieving a faster convergence of the algorithm. Fig. 10 shows the interconnection of the networks to the global control.

The learning memory shown in Fig. 3 is nonvolatile and has all the required training patterns for the training of each of the specialized networks.

The reconfiguration takes place right when a new image must be recognized. Therefore, the architecture has to be modified, and the new training patterns and targets added to the memory. When the training process ends, the memories are updated and the network connections have been already reconfigured so a new recognition process may begin.

According to the research, there are different ways of reconfiguration on a neural network. During the execution time, the number of neurons on the input layer can be modified or enough knowledge can be given to the network by changing the training memory content. Both of these methods explained lead to the recognition of the image.

Depending on the available hardware resources and the applications of the system, a dynamic reconfiguration is possible when the image is part of a class with similar characteristics [35]. Therefore, the reconfiguration of these specialized cooperative networks is made in the control section, increasing their knowledge as new images are recognized.

V. Results

The weights and bias data initially stored in the memory modules were obtained by an online learning phase in the PowerPC, using a back-propagation algorithm. Moreover, simulation software was developed using Matlab and Simulink Neural Network Toolbox. The results obtained in the simulations are described below.

In order to obtain the data of the weights memories, 450 training patterns per class of image (rectangular, circular, and triangular shapes) have been used.

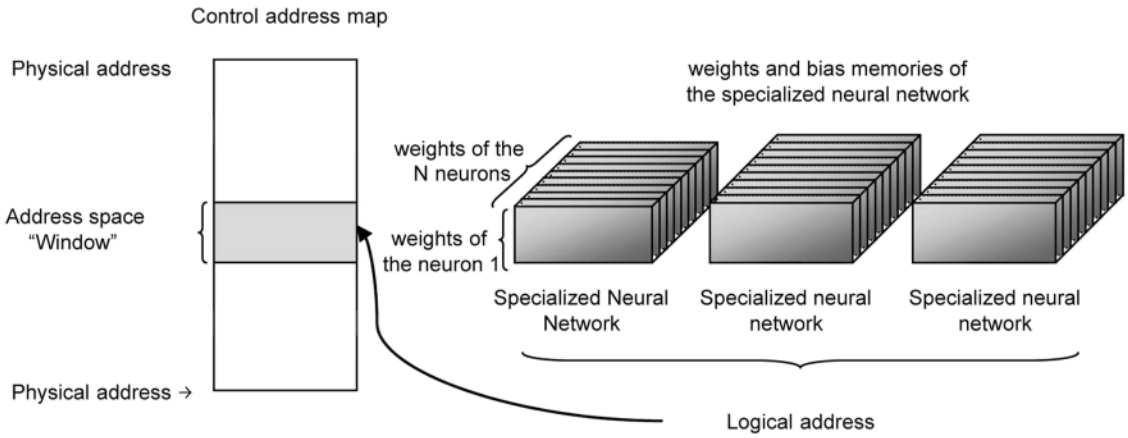

Fig. 10. System memory map.

The training method works in batch mode, which means that once all the patterns have been input, the learning stage updates the weights and biases according to the decreasing moment of the gradient and an adaptive learning scale [28].

The learning strategy followed (offline training with Matlab simulation model, initial online training in the PowerPC and successive relearning online phases) makes possible to functionally validate the network, so that it can be compared with the hardware implementation. Moreover, having the initial weights/biases stored in memory and the network configured makes the successive retraining phases faster.

Some Maltab results are shown in Fig. 11, where the graph shows the stages used for the algorithm to converge with the targets of the parameters [36].

As an example of the application of the system for signal recognition by means of singularities, Fig. 11(a) shows the number of necessary training iterations for one of the networks specialized in recognizing acute-angled corners.

Fig. 11(b) shows the results of the system (uncertainty module) when they present/display 30 images that contain the vectors that are characteristic in a correct sequence corresponding to the recognition of triangular signals.

A region of 6 × 5 (columns*rows) pixels has been used to detect the singularities. The classification mode has been implemented as a series of regions processing. First, a dedicated module detects an special interest region within the image (a PAL field) called *region of interest* (RoI), shown in Fig. 12, sized 60 × 45 pixels, and stores it in the internal embedded RAM memory. Then, successive subzones from this RoI are extracted and sent to the TNN to be processed. Each of these subzones is called *region of detection* (RoD). Therefore, dividing the RoI in 6 × 5 sized-RoDs, results in 10 × 9 data input vectors in one RoI. This is the actual amount of data being processed in each image field, resulting in 90 vectors of 30 pixels each one. This has been accomplished by sweeping the RoI, and by sending each vector of characteristics to the TNN, and by storing the result associated to each region. In this way, probability maps of possible detected singularities are obtained so that the uncertainty stage can decide whether a signal has been detected or not, as shown in Fig. 11(b).

In addition, further simulations in MATLAB, have established that a Q8.16 (fixed-point fractional number binary

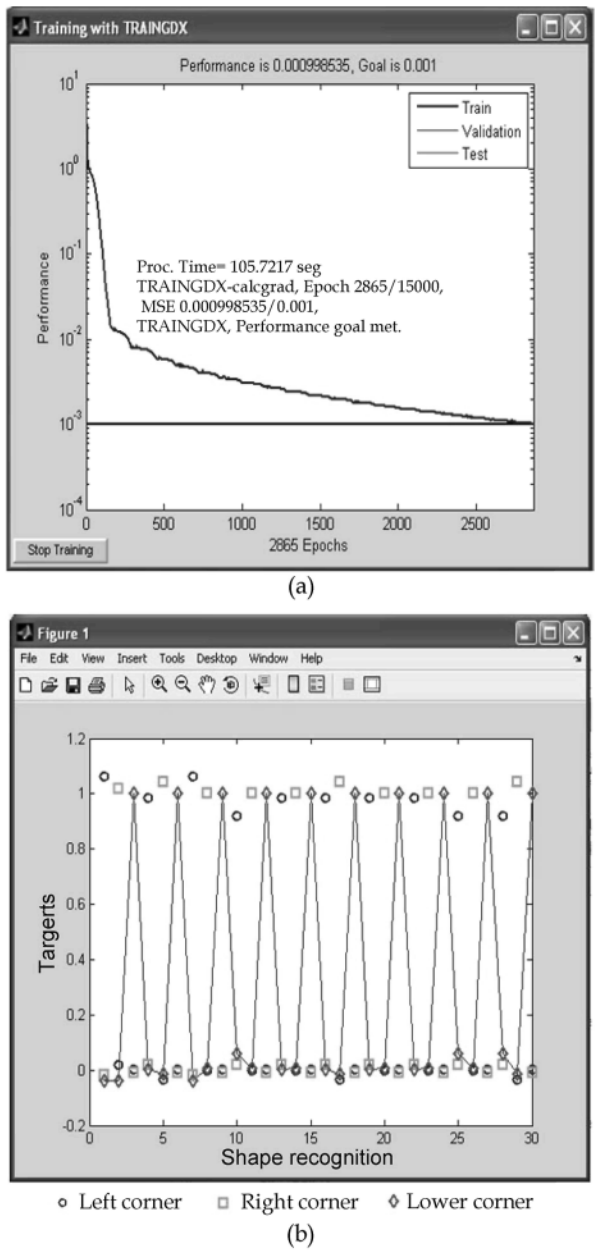

Fig. 11. Training results. (a) Training. (b) Simulation.

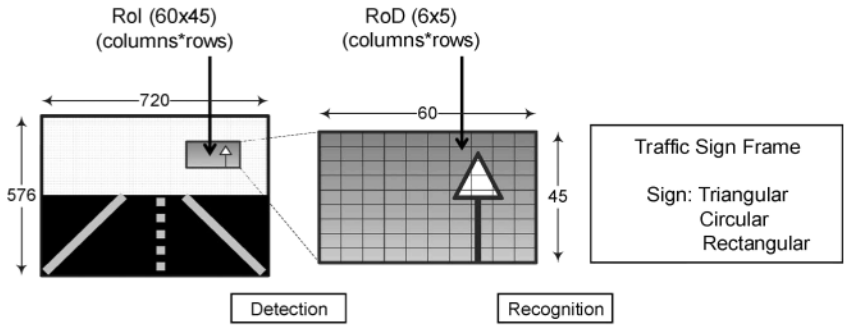

Fig. 12. Traffic sign recognition.

TABLE I
RECOGNITION SYSTEM IMPLEMENTATION RESULTS

Recognition System		***TNN***		
			BKU	RoI Extr.+ RoI Buf.
Logic Elements (LEs)	4,437 / 33,216 (13 %)	*LEs*	2924 / 33,216 (9%)	275 / 33,216
Total Memory Bits	72,416 / 483,840 (15 %)	*Total Mem. Bits*	24480/483,840(5%)	2880 /483,840
M4Ks	47 / 105 (45 %)	*M4Ks*	34 / 105 (33 %)	1 / 105
Frequency	119.32 MHz	*Frequency*	119.32 MHz	

(a) (b)

representation with 8 b to the left of the radix point as the integer part and 16 b to the right as fractional part) format is accurate enough to quantify weights and biases. In comparison with the first approximation made, Q10.22, this reduction in the bit width leads to a reduction in the resources consumed by the network. However, more importantly, a great increase in the maximum operating frequency is also achieved. This is a key factor if we want to enhance the system. This was successfully validated in the FPGA implementation.

As a design premise, we have always had in mind a design for reuse methodology. Therefore, a big effort has been made to specify as many generic hardware modules as possible. For this reason, the architecture and very high speed integrated circuit hardware description language description of the TNN has been improved so that later versions, apart from the basic functionality previously mentioned, allow building N-layer, m-output perceptrons in the easiest and most automated way possible. These features have been incorporated so that we shall be able, in the future, to test the system architecture on larger FPGAs.

Preliminary synthesis (no synthesis effort or optimizations directed to the synthesizer) results for Altera Cyclone EP1C20F400C6 and Cyclone-II EP2C35F672C6 devices have been obtained with the Altera Quartus II (v. 6.0) software package. The proposed architecture (Q8.16) fits in one Cyclone device, but remaining resources, mainly memory, are a bit scarce. Therefore, the system has also been implemented in the Cyclone-II device. Functional and postfitting simulations with Mentor Graphics ModelSim simulation environment show how the real-time restrictions imposed on the system and the functional specifications are met.

Fitting results are shown in Table I(a) for the whole Recognition System (including the TNN, Table I(b), and an image preprocessing stage). The implemented TNN has 30 neurons in the input layer, and 3 in the output layer. Fitting details for the most important blocks of the architecture are also shown.

Table II shows the available resources for the Learning Algorithm implementation on a Xilinx Virtex-II PowerPC. It was coded in C language (400 code lines). Some interesting data are as follows.

1) Learning method: Logsig-Logsig (see Fig. 13).
2) Initial learning rate: Lr $= 0.01$.
3) Maximum learning rate reached during learning: Lrmax $= 0.111437$.
4) Minimum learning rate reached during learning: Lrmin $= 0.001582$.
5) Maximum iterations number: N_ITERATIONS 250.
6) Mean square goal error: TMSE (Total Mean Square Error) > 0.001.
7) Weights and biases randomly initialized between -1 and 1.
8) Convergence reached on iteration 246 (273 real iterations).
9) Final mean-square total error: ECMT $= 0.000994$.
10) Execution time: 3.671 s.
11) Compact Flash memory used to initialize the 450 patterns to train the network.
12) Weights and biases are stored in the double data rate RAM memory (shared memory between the Xilinx and Altera platforms) in the Xilinx board, so that the Altera FPGA can access them during the update phase.

The coding strategy has been directed to minimize the execution time by avoiding the penalization in context switching, so a linear programming model has been followed. Moreover, all the

TABLE II
AVAILABLE RESOURCES FOR THE LEARNING ALGORITHM IMPLEMENTATION

Learning Algorithm (Created by Base System Builder Wizard for Xilinx EDK 8.2 Build EDK_Im.14)	
Target Board	Xilinx XUP Virtex-II Pro Development System Rev C
Family	virtex2p
Device	xc2vp30
Package	ff896
Speed Grade	-7
Processor	PPC 405
Processor clock frequency	300.000000 MHz
Bus clock frequency	100.000000 MHz
Debug interface	FPGA JTAG
Data Cache	16 KB
Instruction Cache	16 KB
On Chip Memory	208 KB
Total Off Chip Memory	512 MB
	- DDR_SDRAM_64Mx64 Dual Rank = 256 MB
	- DDR_512MB_64Mx64_rank2_row13_col10_cl2_5 = 256 MB

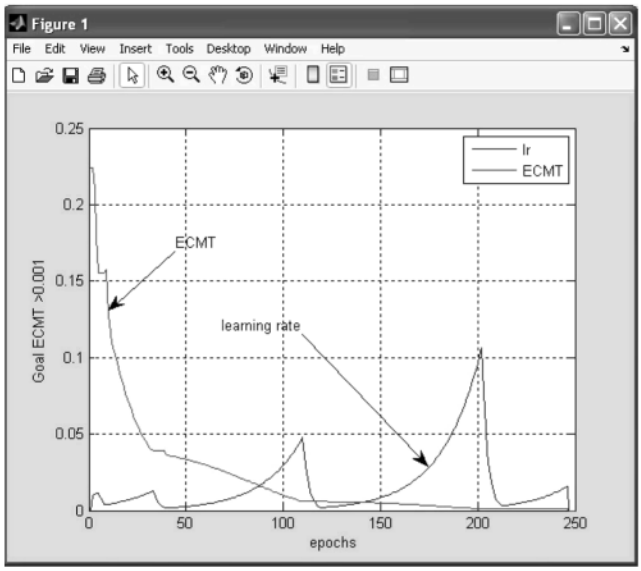

Fig. 13. Learning rate.

loops have been unrolled to take advantage of the five pipeline stages of the PowerPC. The training algorithm execution takes about 3 min.

VI. CONCLUSION

We have proposed and designed a new hardware architecture for a neural network system based on specialized TNNs for image recognition. One of the most important features of TNNs is their online learning capability. Moreover, they follow a cooperative working strategy among them, in order to solve complex recognition problems. As an example of this system application, TNNs trained to identify the mentioned shapes singularities may be used to detect traffic signs and/or pedestrians in a driving scenario. Therefore, advanced driver assistance systems may perfectly use this kind of system. In addition, an autonomous robot, and, in general, any kind of autonomous navigation system, may eventually benefit from the use of this cooperative, extremely fast, and reliable TNNs to make their navigation safer by detecting the contour of different possible objects that surround them [37], [38].

The next step will be porting the Altera implementation of the network to the Virtex II-Pro device, together with the PowerPC responsible of the online training. The reason to port the whole system to the Virtex device is the possibility to use the couple of hard-embedded PowerPC processors so that the processing power is increased and so, the back-propagation algorithm can be accelerated, taking into account the segmentation and parallelism degree in each one of the algorithm stages. Eventually, the learning algorithm will also be transformed to hardware for further performance improvements, so we need a more powerful device. In this scenario, a hard-embedded processor may at first be considered as irrelevant, but we predict a further enhance of the system that may require intensive algorithmic stages for decision taking situations.

The last goal of these efforts is trying to embed more intelligence in the actual embedded systems, by scaling and transforming into hardware some of the typical artificial intelligence algorithmic tools.

REFERENCES

[1] A. de la Escalera, L. E. Moreno, M. A. Salichs, and J. M. Armingol, "Road traffic sign detection and classification," *IEEE Trans. Ind. Electron.*, vol. 44, no. 6, pp. 848–859, Dec. 1997.

[2] A. de la Escalera, L. Moreno, E. A. Puente, and M. A. Salichs, "Neural traffic sign recognition for autonomous vehicles," in *Proc. IEEE Int. Conf. Ind. Electron., Control Instrum.*, 1994, vol. 2, pp. 841–846.

[3] T. Theocharides, G. Link, N. Vijaykrishnan, M. J. Invin, and V. Srikantarn, "A generic reconfigurable neural network architecture as a network on chip," in *Proc. IEEE Int. SOC Conf.*, 2004, pp. 191–194.
[4] S. Estable, J. Schick, F. Stein, R. Janssen, R. Ott, W. Ritter, and Y. J. Zheng, "A real time traffic sign recognition system," in *Proc. Intell. Vehicles Symp.*, 1994, pp. 213–218.
[5] J. Torresen, J. W. Bakke, and L. Sekanina, "Efficient recognition of speed limit signs," in *Proc. IEEE Int. Conf. Intell. Transp. Syst.*, 2004, pp. 652–656.
[6] V. Moreno, A. Ledezma, and A. Sanchis, "A static images based-system for traffic signs detection," in *Proc. IASTED Int. Conf. Artif. Intell. Appl.*, 2006, pp. 445–450.
[7] G. Adorni, V. D'Andrea, G. Destri, and M. Mordonini, "Shape searching in real word images: A CNN based approach," in *Proc. 4th IEEE Int. Workshop Cellular Neural Netw. Appl.*, 1996, pp. 213–218.
[8] J. Alarcón, R. Salvador, F. Moreno, and I. López, "A new real-time hardware architecture for road line tracking using a particle filter," in *Proc. 32nd Annu. IEEE IECON*, Paris, France, 2006, pp. 736–741.
[9] I. López, R. Salvador, J. Alarcón, and F. Moreno, "Architectural design for a low cost FPGA-based traffic signal detection system in vehicles," in *Proc. SPIE*, Gran Canaria, Spain, 2007, vol. 6590, p. 659 00M.
[10] M. Xiaobin, J. Lianwen, S. Dongsheng, and Y. Junxun, "A mixed parallel neural networks computing unit implemented in FPGA," in *Proc. IEEE Int. Conf. Neural Netw. Signal Process.*, Nanjing, China, Dec. 2003, pp. 324–327.
[11] M. Avogadro, M. Bera, G. Danese, F. Leporati, and A. Spelgatti, "The Totem neurochip: An FPGA implementation," in *Proc. 4th IEEE Int. Signal Process. Inf. Technol.*, 2004, pp. 461–464.
[12] S. Bridges, M. Figueroa, D. Hsu, and C. Diorio, "A reconfigurable VLSI learning array," in *Proc. 31st Eur. Solid-State Circuit Conf.*, 2005, pp. 117–120.
[13] T. Fukuda, T. Shibata, M. Tokita, and T. Mitsuoka, "Neuromorphic control: Adaption and learning," *IEEE Trans. Ind. Electron.*, vol. 39, no. 6, pp. 497–503, Dec. 1992.
[14] E. Monmasson and M. N. Cirstea, "FPGA design methodology for industrial control systems—A review," *IEEE Trans. Ind. Electron.*, vol. 54, no. 4, pp. 1824–1842, Aug. 2007.
[15] N. M. Botros and M. Abdul-Aziz, "Hardware implementation of an artificial neural network using field programmable gate arrays (FPGA's)," *IEEE Trans. Ind. Electron.*, vol. 41, no. 6, pp. 665–667, Dec. 1994.
[16] M. A. Figueiredo and C. Gloster, "Implementation of a probabilistic neural network for multi-spectral image classification on an FPGA based custom computing machine," in *Proc. Vth Brazilian Symp. Neural Netw.*, 1998, pp. 174–178.
[17] H. Hikawa, "Implementation of simplified multilayer neural networks with on-chip learning," in *Proc. IEEE Int. Conf. Neural Netw.*, 1995, vol. 4, pp. 1633–1637.
[18] S. B. Yun, Y. J. Kim, S. S. Dong, and C. H. Lee, "Hardware implementation of neural network with expandible and reconfigurable architecture," in *Proc. IEEE Int. Conf. Neural Inf.*, 2002, vol. 2, pp. 970–975.
[19] B. Pino, F. J. Pelayo, J. Ortega, and A. Prieto, "Design and evaluation of a reconfigurable digital architecture for self-organizing maps," in *Proc. Int. Conf. Microelectron. Neural, Fuzzy Bio-Inpired Syst.*, 1999, pp. 395–402.
[20] D. Hammerstrom, "A VLSI architecture for high-performance, low cost, on-chip learning," in *Proc. IJCNN Int. Conf. Neural Netw.*, 1990, vol. 2, pp. 537–544.
[21] S. Vitabile, A. Gentile, G. B. Dammone, and F. Sorbello, "Multi-layer perceptron mapping on a SIMD architecture," in *Proc. 12th IEEE Workshop Neural Netw. Signal Process.*, 2003, pp. 667–675.
[22] I. Lopez, *Visión por singularidades*. Madrid, Spain: Auton. Syst. Lab. (ASLab), Universidad Politecnica de Madrid (UPM), 2001. Internal Publication.
[23] L. Priese, R. Lakmann, and V. Rehrmann, "Ideogram identification in a realtime traffic sign recognition system," in *Proc. IEEE Intell. Vehicles*, Sep. 25–26, 1995, pp. 310–314.
[24] Y. E. Krasteva, E. de la Torre, and T. Riesgo, "Partial reconfiguration for core relocation and flexible communications," in *Proc. Reconfigurable Commun.-Centric SoC*, Montpellier, France, pp. 91–97.
[25] J. L. Beuchat, J. O. Haenni, and E. Sanchez, "Hardware reconfigurable neural networks," in *IPPS, SPDP Worshops*, 1998, pp. 91–98. [Online]. Available: citeseer.ist.psu.edu/beuchat98hardware.html
[26] J. A. Starzyk, Z. Zhen, and L. Tsun-Ho, "Self-organizing learning array," *IEEE Trans. Neural Netw.*, vol. 16, no. 2, pp. 355–363, Mar. 2005.
[27] J. G. Eldredge and B. L. Hutchings, "Density enhancement of a neural network using FPGAs and run-time reconfiguration," in *Proc. IEEE Workshop FPGAs for Custom Mach.*, Apr. 10–13, 1994, pp. 180–188.
[28] M. T. Hagan, H. B. Demuth, and M. Beale, *Neural Network Design*. Stamford, CT: Thomson Learning, 1996.
[29] B. Kröse and P. Van der Smagt, *An Introduction to Neural Networks*, 8th ed. Amsterdam, The Netherlands: Univ. Amsterdam, 1996.
[30] H. M. Tai, J. Wang, and K. Ashenayi, "A neural network-based tracking control system," *IEEE Trans. Ind. Electron.*, vol. 39, no. 6, pp. 504–510, Dec. 1992.
[31] M. A. Hannan Bin Azhar and K. R. Dimond, "Design of an FPGA based adaptive neural controller for intelligent robot navigation," in *Proc. IEEE Euromicro Symp. Digital Syst. Des.*, 2002, vol. 2, pp. 283–290.
[32] Y. Taright and M. Hubin, "FPGA implementation of a multilayer perceptron neural network using VHDL," in *Proc. 4th Int. Conf. Signal Process.*, 1998, vol. 2, pp. 1311–1314.
[33] J. J. Blake, L. P. Maguire, T. M. McGinnity, and L. J. McDaid, "Using Xilinx FPGAs to implement neural networks and fuzzy systems," in *IEE Colloq. Neural Fuzzy Syst.: Design Hardware Appl. (Digest No. 1997/133)*, pp. 1/1–1/4.
[34] A. Perez-Uribe and E. Sanchez, "Implementation of neural constructivism with programmable hardware," in *Proc. Int. Symp. Neuro-Fuzzy Syst.*, 1996, pp. 47–54.
[35] M. S. Obaidat and D. T. Macchiarolo, "An online neural network system for computer access security," *IEEE Trans. Ind. Electron.*, vol. 40, no. 2, pp. 235–242, Apr. 1993.
[36] MATLAB, *The Language of Technical Computing*. Version 7.4.0.287 (R2007a).
[37] P. Vadakkepat, P. Lim, L. C. De Silva, L. Jing, and L. L. Ling, "Multimodal approach to human-face detection and tracking," *IEEE Trans. Ind. Electron.*, vol. 55, no. 3, pp. 1385–1393, Mar. 2008.
[38] T. Ozaki, T. Suzuki, T. Furihashi, S. Okuma, and Y. Uchikawa, "Trajectory control of robotic manipulators using neural networks," *IEEE Trans. Ind. Electron.*, vol. 38, no. 3, pp. 195–202, Jun. 1991.

Félix Moreno was born in Valladolid, Spain, in 1959. He received the M.Sc. and Ph.D. degrees in telecommunication engineering from the Universidad Politécnica de Madrid (UPM), Madrid, Spain, in 1986 and 1993, respectively.

He is currently an Assistant Professor of electronics with the UPM. He has published a large number of papers in those fields and has participated and acted as main researcher in several national- or European Union-funded projects. His research interests are focused on evolvable hardware, high-performance reconfigurable and adaptive systems, hardware embedded intelligent architectures, and digital signal processing systems.

Jaime Alarcón received the B.S. degree in electrical and mechanical engineering and the M.S. degree in engineering from the Universidad Nacional Autónoma de México, Mexico City, Mexico, the Master's degree in computer sciences from the Instituto Tecnologico de Estudios Superiores de Monterrey, Toluca, México, and the Ph.D. degree in electronics engineering from the Universidad Politecnica de Madrid, Madrid, Spain.

He is a Full-Time Professor of electronics engineering with the TEC de Monterrey Campus Toluca, Toluca, México, where he also served as Professor and Manager of the electronics and control department. His research interests are based on neural networks, parallelism, hardware embedded architectures, and real-time systems.

Rubén Salvador received the B.Sc. degree in telecommunication engineering from the Universidad Politecnica de Madrid (UPM), Madrid, Spain, in 2001, the M.Sc. degree in electrical and electronic engineering from the Universidad de Alcala, Madrid, in 2004, and M.Sc. degree in industrial electronics from the UPM, in 2007, where he is currently working toward the Ph.D. degree in the Centro de Electrónica Industrial.

From January 2005 to October 2006, he worked as a Researcher with the Intelligent Vehicle Systems Division, University Institute for Automobile Research, UPM. His research interests include evolvable hardware, high-performance reconfigurable and adaptive systems, hardware embedded intelligent architectures, and digital signal processing systems.

Teresa Riesgo (M'96) was born in Madrid, Spain, in 1965. She received the M.Sc. and Ph.D. degrees in electrical engineering from the Universidad Politecnica de Madrid (UPM), Madrid, in 1989 and 1996, respectively.

Since 2003, she has been a Full Professor of electronics with the UPM. She has published a large number of papers in those fields and has participated and acted as Main Researcher in several European Union-funded projects. She is currently the Director of the Centro de Electrónica Industrial, UPM. Her research interests are focused on embedded-system design, wireless-sensor networks, configurable systems, and power estimation in digital systems.