EURASIP Journal on Applied Signal Processing 2005:7, 993–1004

Object Recognition System-on-Chip Using the Support Vector Machines

Roberto Reyna-Rojas
Laboratory for Analysis and Architecture of Systems (LAAS), CNRS, 7 avenue du Colonel Roche, 31077 Toulouse Cedex 4, France
Email: roberto.reyna@ieee.org

Dominique Houzet
The Rennes Institute of Electronics and Telecommunications (IETR) (UMR CNRS 6164), INSA, 20 avenue des Buttes de Coësmes, 35053 Rennes Cedex, France
Email: houzet@insa-rennes.fr

Daniela Dragomirescu
Laboratory for Analysis and Architecture of Systems (LAAS), CNRS, 7 avenue du Colonel Roche, 31077 Toulouse Cedex 4, France
Email: daniela@laas.fr

Florent Carlier
The Rennes Institute of Electronics and Telecommunications (IETR) (UMR CNRS 6164), INSA, 20 avenue des Buttes de Coësmes, 35053 Rennes Cedex, France
Email: florent.carlier@univ-lemans.fr

Salim Ouadjaout
The Rennes Institute of Electronics and Telecommunications (IETR) (UMR CNRS 6164), INSA, 20 avenue des Buttes de Coësmes, 35053 Rennes Cedex, France
Email: salim.ouadjaout@insa-rennes.fr

Received 16 September 2003; Revised 6 June 2004

The first aim of this work is to propose the design of a system-on-chip (SoC) platform dedicated to digital image and signal processing, which is tuned to implement efficiently multiply-and-accumulate (MAC) vector/matrix operations. The second aim of this work is to implement a recent promising neural network method, namely, the support vector machine (SVM) used for real-time object recognition, in order to build a vision machine. With such a reconfigurable and programmable SoC platform, it is possible to implement any SVM function dedicated to any object recognition problem. The final aim is to obtain an automatic reconfiguration of the SoC platform, based on the results of the learning phase on an objects' database, which makes it possible to recognize practically any object without manual programming. Recognition can be of any kind that is from image to signal data. Such a system is a general-purpose automatic classifier. Many applications can be considered as a classification problem, but are usually treated specifically in order to optimize the cost of the implemented solution. The cost of our approach is more important than a dedicated one, but in a near future, hundreds of millions of gates will be common and affordable compared to the design cost. What we are proposing here is a general-purpose classification neural network implemented on a reconfigurable SoC platform. The first version presented here is limited in size and thus in object recognition performances, but can be easily upgraded according to technology improvements.

Keywords and phrases: parallel architecture, pattern recognition, support vector machines, hardware design language, systems-on-programmable-chip and system-on-chip platforms.

1. INTRODUCTION

This work relates to machine vision but is considered under the angle of the hardware design and integration. This work will be centered on specific signal processing circuits. We have chosen the SVM neural network algorithm as our data classification algorithm.

Artificial neural networks became a very powerful tool and are used for feature extraction and for high-level decisions. They are founded on experimental data analysis and processing. They are the basis of expert systems and thus used when there is an insufficient knowledge of the studied process. It will be also possible, as mentioned in the abstract,

to use them when the design time is shortened as it is the case with time-to-market constraints. The neural networks by themselves represent a significant research subject in the scientific and technological world since a few tens of years. Theoretical bases, performances, architectures, applications, and hardware implementations are some of the studied axis [1].

A machine-vision design relates also to the hardware part of a system. For some particular applications, hardware design goes from the study and the design of image sensors and optics to computing units. This work is rather centered on the computing units dedicated to application algorithms, using a standard camera for image acquisition. In commercial systems, we frequently find architectures using traditional processors, which provide the necessary performances to applications. We also can find architectures with specialized digital signal processing circuits (DSP), which have suitable arithmetic units for the necessary precision. Nevertheless, the regularity of image processing and neural network algorithms cannot be completely exploited by these types of architectures. Parallel architectures are best adapted for hardware implementation of vision systems and neural calculations due to their ability to exploit the parallel nature of algorithms.

The growing scale of integration has allowed designers to include in the same chip several parts of a system and even the entire system. Systems-on-chip (SoC) is one of the latest ideas in system integration. Circuits cannot be designed in a classical way because they are more complex and different functions (subsystems) are being integrated. Technology allows more flexible architectures: a larger number of integrated gates, less power consumption, higher speeds, bigger and faster integrated memories, processors cores, communications interfaces, and so forth. Object recognition system-on-chip is a natural perspective in the machine perception domain.

In Section 2 of this paper, we present the basic idea of the SVM, in particular for classification. In Section 3, we explain the algorithm complexity and the software performances of the SVM method. We briefly present neural architectures in Section 5 and some application results in Section 6. In Sections 7 and 8, we will give the details of our proposed architecture of the SoC platform solution and we will end our paper with the conclusion and perspectives.

2. THE SUPPORT VECTOR MACHINES

Twenty years ago, the neural networks knew a very significant importance in scientific and engineering worlds. Nowadays, industrial products are offered on the market with real success even if we do not have the associated physical model within the automation or the diagnosis. It is necessary to consider the neural networks as a tool for building an empirical model with what that supposes of inaccuracy and risk for the application. The theory of the statistical learning became more interesting with new results in generalization and with the proposal of the SVM model. Vapnik in the AT&T Bell laboratories proposed the theory of the statistical learning [2, 3]. We will very briefly present this theory in order to introduce the generalization function. The details of the theory can be consulted in [2].

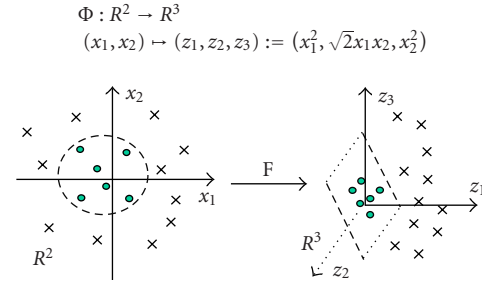


Figure 1: Kernel functions are used to transform the input space into feature space where the optimal hyperplane is constructed.

The theory of the SVM

The support vector machine model is the most recent proposition on neural network structures. This model is based on the statistical learning theory. The support vector machine model consists in a transformation of the input vectors X in a space of higher dimension Z through a nonlinear transformation, selected a priori. It is in this new space Z that we can build an optimal hyperplane [2]. For the particular case of pattern recognition, the SVMs make a distinction of two classes by finding a decision surface constructed from certain points of the entire learning database, called support vectors [4].

Vapnik proposes a representation of an SVM in the form of one-hidden-layer neural network whose number of cells is equal to the number of "support vectors," and not to the dimension of the space of the internal representations, as we could have supposed it initially. In this manner, the number of neurons is obtained in an automatic way with the resolution of a quadratic problem. The support vectors are the input vectors x_i for which equality $y_i(w_0x_i + b_0) = 1$ holds. Concretely, they are the closest points to the optimal hyperplane. For all other examples, there is thus a factor $\alpha = 0$ that eliminates them from the solution. We thus know that the decision function is calculated from the examples that are on the margin. In the nonlinear case, it is enough to replace the scalar products $(x \cdot x_i)$ by kernels $k(x, x_i)$. The kernel functions were proposed to build nonlinear algorithms from linear algorithms by calculating the inner product not in the input space but in the feature space. Figure 1 shows this transformation.

The three most common options for the selection of the kernel function of the SVM method are the polynomial, RBF, and sigmoid neural networks. The sigmoid neural network kernel function option was rejected in this work because of the difficulty of hardware implementation. Moreover, in the literature, the performances obtained with this kernel function are less interesting than those obtained with the two others. The results on the applications (cf. Section 4) showed that, with the polynomial kernel function, we obtain

a solution, for different databases, with the minimum number of support vectors. In terms of generalization, we observed, particularly in the first application, that the best performances were also obtained with the polynomial kernel.

3. COMPLEXITY AND PERFORMANCES

The general equation of the SVM generalization function for classification is

$$f(x,\alpha) = \operatorname{sign}\left(\sum_{\text{Support Vectors}} y_i\alpha_i k(x_i,x) + b\right), \quad (1)$$

where

(i) $y_i\alpha_i = w_i$ are the network weights,
(ii) x_i are the support vectors of the solution,
(iii) b is the threshold of the function,
(iv) $k(x,x_i)$ is the kernel function.

As we can see, the solution is the sign of the sum, which is the generalization function for a two-class classification.

In our case, the kernel function is the polynomial function of degree d:

$$k(x,y) = (x\cdot y + c)^d. \quad (2)$$

The principal parameter of the polynomial kernel function is the polynomial degree. We take as a priori choice a polynomial of degree 2 (a higher degree implied the use of wider data buses in the hardware implementation).

3.1. Complexity

We suppose that the image size is $\mathbf{t}_m \times \mathbf{t}_m$ and that $\mathbf{t}_b \times \mathbf{t}_b$ is the detection window size. $\mathbf{t}_b^2$ is thus the number of pixels to be processed by the window of classification. Here we consider a decision function of SVM with a polynomial kernel of degree d:

$$f(x,\alpha) = \operatorname{sign}\left(\sum_{\text{Support Vectors}} y_i\alpha_i\left[(x_i\cdot x)+1\right]^d + b\right). \quad (3)$$

If we write $w_i = y_i\alpha_i$, we have

$$f(x,\alpha) = \operatorname{sign}\left(\sum_{\text{Support Vectors}} w_i\left[(x_i\cdot x)+1\right]^d + b\right). \quad (4)$$

To make the classification of all the windows of pixels of one 512×512 image, with no sweeping, and an 8×8 detection window, we have 64×64 $(\mathbf{t}_m/\mathbf{t}_b)^2$ windows to process. Each window (or input vector) requires $\mathbf{t}_b^2$ operations (operation = multiplication + addition) for the scalar product of the kernel function $(x\cdot x_i)$ and d multiplications for power operation, which we also consider as one operation for simplicity. We have then

$$\mathbf{t}_b^2 + d + 1 \text{ operations per support vector.} \quad (5)$$

Table 1: Summary of the algorithm complexity.

Algorithm	Number of operations
SVM	$N\times\mathbf{t}_m^2$
SVM sweeping window	$(\mathbf{t}_b/p)^2\times N\times\mathbf{t}_m^2$
Convolution	$M^2\times\mathbf{t}_m^2$

The additional operation is due to the multiplication between the weight w_i and the result of the polynomial and the addition of the threshold $\mathbf{b}$. Let N be the number of support vectors obtained during learning; we will then have

$$N\times(\mathbf{t}_b^2 + d + 1) \text{ operations per block.} \quad (6)$$

For $(\mathbf{t}_m/\mathbf{t}_b)^2$ windows per image, we obtain

$$N\times(\mathbf{t}_b^2 + d + 1)\times\left(\frac{\mathbf{t}_m}{\mathbf{t}_b}\right)^2 \text{ operations per image.} \quad (7)$$

By making a simplification and knowing that in general, $\mathbf{t}_b^2 \gg d+1$, we thus have $N\times\mathbf{t}_m^2$ operations per image.

That means that the number of operations to be calculated depends on the image size and on the number of support vectors. The size of the window thus does not have a significant influence on the complexity of the algorithm. Nevertheless, this size will represent a fundamental factor during the material implementation because it will be used to dimension part of the circuit.

Now, if we use a sweeping classification window over the image, we will classify pixels several times. In this case, there will be more windows to analyze per image: $(\mathbf{t}_m/p)^2$, where p is the number of sweeping pixels (can also be seen as the classification resolution). For example, for $p=2$, we move in the image with a step of 2 pixels at a time, horizontally and vertically. We then get

$$N\times(\mathbf{t}_b^2 + d + 1)\times\left(\frac{\mathbf{t}_m}{p}\right)^2 \approx \left(\frac{\mathbf{t}_b}{p}\right)^2\times N\times\mathbf{t}_m^2 \quad (8)$$

operations per image.

In the case of an 8×8 detection window and a sweeping step of 2 pixels, we will make 16 times more calculations than without sweeping. The advantage of using sweeping would be to increase the image sampling and to classify several times each pixel or window of pixels and thus to obtain a more robust decision, and also to increase at the same time the localization precision. The complexity for a traditional image processing algorithm like filtering by a convolution direct method depends on the size of the convolution mask ($M\times M$ for example) and on the size of the processed image, therefore the number of operations is given by

$$M^2\times\mathbf{t}_m^2 \text{ operations by image.} \quad (9)$$

Table 1 summarizes the algorithm complexity analysis. Applying a convolution mask to an image is less expensive in computing requirements than the other algorithms if the size of the mask M is higher than 9. Nevertheless, applying

Table 2: Execution times for diferent image sizes: 16×16 window size, 88 support vectors.

Image size	Execution time		Execution time Sweeping window	
	Estimated	Measured	Estimated	Measured
128×128	0.7 s	0.6 s	2.2 s	2.7 s
256×256	2.5 s	2.7 s	9.2 s	10.8 s
512×512	10.2 s	11.0 s	37.0 s	43.9 s

the convolution mask is only the first step to solve the problem of object detection and localization.

In general, if we use a classical method for object recognition, the complexity of the system will be the addition of the complexity of each subsystem. It will also depend on different parameters of the processed image, for example, edges density, line density, and the ratio between the object and the image size. For the SVM method, the complexity depends only on a priori chosen parameters.

3.2. Performances

We carried out some measurements of execution times. As we have shown, the number of operations and the computing time increase proportionally to the number of support vectors. We thus found the main disadvantage of the support vector machine method: the number of support vectors. This number is automatically obtained during learning; we cannot control this parameter without modifying the generalization performances.

These measurements of execution times were made on a Sun Microsystems Ultra 5 Workstation.

For the estimation of the computing time, we obtained that a multiplication-addition operation is executed in 470 nanoseconds. We obtained this time from a program carrying out a loop of 10^6 iterations. In this loop as in the software implementation of the function of generalization of the SVM, we used the mathematical function pow$(\cdot)$. Estimated times are slightly larger than measured times. This is due to the use of the indices in the estimation program. Table 2 shows some results.

3.3. Learning performances

The learning algorithm uses a decomposition method to increase the learning performance and to reduce the necessary resources of the machine on which we execute the learning algorithm, in particular, memory resources. This algorithm calls the generalization function and supposes that we can define a working set (vectors or examples) B such as $|B| \leq L$ (L is equal to the number of examples or vectors of all the learning database, and $|B|$ the number of B elements). This set is sufficiently large to contain all the support vectors ($\alpha_i > 0$), but sufficiently small so that the hardware platform (PC, workstation, etc.) can handle and optimize them by using the quadratic optimization algorithm.

The decomposition technique can be written in the following manner.

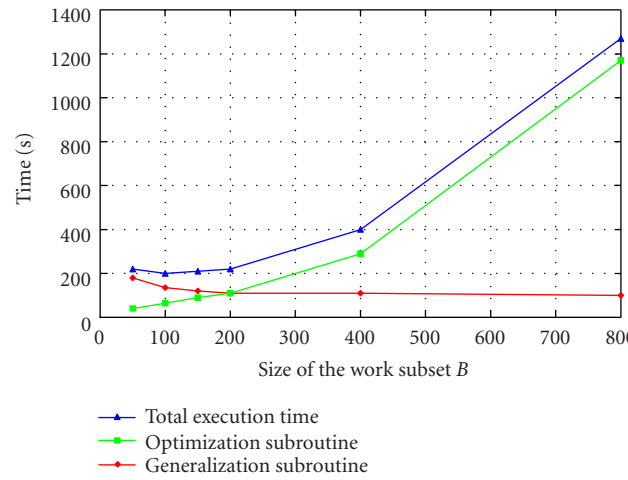


Figure 2: Execution time of the learning program and the optimization and generalization subroutines of the SVM method, obtained by using a database of 4096 examples of dimension 64.

(1) Choose in a random way $|B|$ points of the database.
(2) Resolve the subproblem defined by the elements in B.
(3) Repeat the three steps while there exists a $j \in \mathbb{N}$ such as $g(x_j) \cdot y_j < 1$ (which corresponds to a bad classification), where

$$g(x_j) = \sum_{p=1}^{l} y_p \alpha_p K(x_j, x_p) + b. \tag{10}$$

The algorithm, at each iteration, improves the objective (optimization) function and is not, in this sense, recursive. Since the objective function is limited, the algorithm converges towards the optimal solution in a finite number of iterations [5].

The function $g(x_j)$ is in fact the SVM generalization function; and for instance, if we are able to reduce two orders of magnitude, the execution time of this part of the algorithm will improve the learning performances and we could have a real-time learning algorithm.

We can observe the experimental results of the execution time of the learning algorithm according to the size of the working subset B in Figure 2. The learning process is clearly accelerated with this decomposition method. According to these simulation results, for a real-time learning system and for subsets of average sizes, it would be necessary to increase the performances of the execution of the quadratic optimization algorithm. We can also observe in Figure 2 that the execution time of the SVM generalization function is practically constant, which is approximately 100 seconds. This is because the calculation of $g(x_j)$ is made for all $\mathbf{j}$ of the database and thus does not depend on B. For B lower than 200, the execution time of the learning algorithm is practically dominated by the generalization subroutine.

As we also can see, the software execution times are prohibited for real-time applications. This is the reason of the hardware implementation. We are now presenting some of

the results on one of the three tested applications and then we are going to detail the architecture at different levels.

4. APPLICATIONS

The excellent performances of the SVM for classification problems were very attractive from the beginning of their proposal. This is true especially if we consider that the method can be applied directly on pixel values, it does not need to take into account any other a priori problem knowledge, and "a permutation of the images by a fixed transformation does not modify the SVM classifier performances" [6].

The performance analysis of the SVM methods on databases used as "benchmarks" by the scientific community was already reported in literature [2, 7]. Other evaluations were made on synthetic databases [8]. The principal interest of our contribution is to study this method for real-life applications (matrix barcodes detection, face detection in an automobile cockpit, and the white lines detection). We have found that the SVM method makes, possible to build very powerful classifiers (polynomial, RBF, or perceptron).

4.1. Detection and localization of matrix barcodes

Barcodes are essential as a product identification, either during manufacturing or during marketing. The market requirements made very important the fine resolution of questions like reading robustness under very diverse conditions. The effectiveness of barcodes is so interesting that the vendors would wish to be able to put more information on them. A linear barcode, for example EAN13 code, can code 11 characters (numerical 0–9); this code is generally used like reference for a product index. The aim of matrix barcodes is to be able to code more than 2000 alphanumeric characters, and to thus be able to have product information like their price and their principal features. That supposes to evolve from a one-dimensional code to a two-dimensional code; and two-dimensional codes suppose image processing and recognition.

This study was made with the collaboration of Intermec Company. Intermec provided a base of 78 images with different types of matrix barcodes and various image sizes. The study was based on the DataMatrix code. We have also shown results of generalization on other types of codes. Each pixel value is coded on eight bits, that it, in 256 gray levels from 0 to 255.

The images show different scenarios like projective deformations, different image backgrounds, different scales, and so forth. For this application, we find the object by segmenting the image and not by finding directly the whole object, that is, we benefit from the texture regularity of matrix barcodes to locate them. In [9], the author proposes, for the localization and the automatic reading of matrix barcodes, to use the texture to validate the different zones found by the localization algorithm. The objective for this first application is thus to learn texture from a matrix barcode DataMatrix and to make a localization of these codes in new images through image segmentation.

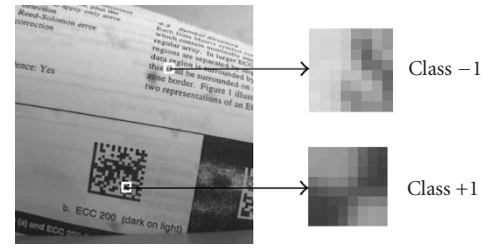


Figure 3: Definition of the two classes for matrix barcodes detection.

Table 3: Number of support vectors found during learning for different kernels and values of parameter C.

Kernel	Degree	Value of C			
		10	200	500	5000
Linear	—	491	547	620	1320
Polynomial	2	316	310	321	320
Polynomial	3	333	343	325	311
Polynomial	4	341	310	311	311
RBF	2	385	333	312	304

Databases creation is a delicate task for the methods that use supervised learning algorithms. The solution of the neural network will depend exclusively on the examples of the learning database. Since the SVM method is also based on learning from examples, a given "optimal" learning database provides an "optimal" solution.

In this application, we feed the learning algorithm with examples of the "positive" parts of the image (a matrix barcode), and with other textures (text, images, etc.) as "negative" examples. Two classes are thus defined (see Figure 3): a block of pixels with the texture of the matrix barcode (class +1) and a block of pixels with a different texture (class −1). Two detection window sizes were tested: 8×8 pixels and 16×16 pixels.

We present the learning results over one database and the respective result in generalization. The first database was created from the image shown in Figure 3. In Table 3, we show learning results with a database of 4096 input vectors of dimension 64 (8×8 pixels) with 240 positive examples. The number of support vectors is indicated in Table 3 for different kernel functions and different values of the penalization parameter C.

We created a test database from three different images. This test database consists of 12 288 examples, including 10% of positive examples. In generalization, we obtained 91.8% of good classifications (positive or negative), 3.8% of no detection (when the example is positive and the result of generalization is negative), and 4.2% of false detection (when the example is negative and the result of generalization is positive).

For a second test database with 10 465 examples, including 25% positive, we had 85% of success or good classifications, 11% of no detection, and 4% of false alarm. The fact of

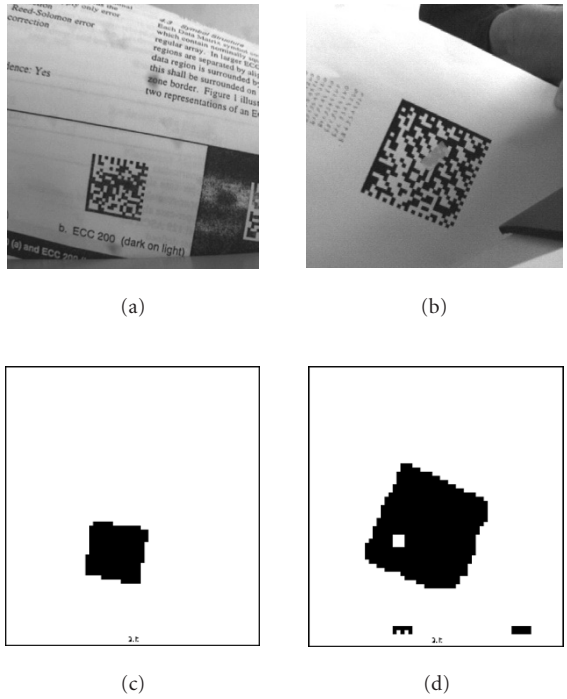


Figure 4: Image segmentation results using the SVM as detection system. The window size is 8×8 pixels. A postprocessing algorithm is used in order to erase the bad classifications of the SVM.

having a relatively small percentage of false alarms compared to the number of no detection led us to define a postprocessing module based on a morphological processing for this particular application. In the images of Figure 4, we show some qualitative results of detection, that is, we show two examples of the output binary images.

We seek to use the best solution with a minimal number of support vectors. These results were obtained with the second-degree polynomial kernel function solution and with the penalization parameter C = 200. The first image shows the result of the test since the learning database was created using this image. More detailed information can be found in [10].

5. PARALLEL NEURAL ARCHITECTURES

The regularity of image processing and neural network algorithms encourages the use of parallel VLSI circuits. Parallelism is an intrinsic notion of the neural networks, which are regarded as massively parallel systems [11]. In spite of the enormous computing power obtained with new sequential processors, it is possible that these types of processors are not sufficient for real-time applications. There are some solutions with neural networks, which use classical sequential processors, for example, the optical character recognition (OCR) algorithms, whose performances are acceptable for applications that do not require a real-time operation.

A significant number of analog implementations were proposed, exploiting the biological origin of neural networks, which illustrates the use of individual simple cells but interconnected by a network and functioning in a massively parallel way. In the particular case of the integrated artificial retinas, the use of analog circuits is a choice impossible to avoid, because we want to be able to bring processing as near as possible to the photosensitive circuit and to be able to manage the interconnections more easily (each pixel interacts with its closer neighbors) [12, 13].

Many neural implementations in numerical integrated circuits have been proposed. The finality of these circuits is to be used within traditional workstations like neural coprocessors, in acquisition and signal processing cards, in order to make more intelligent sensors, or to be used as specialized parallel-processing machines. They are generally dedicated to a single neural model, and all do not propose a learning integrated procedure [14]. There are thus several types of neural systems.

(1) Application-specific architectures implement a model, a topology, and a set of weights, mostly by analog means.

(2) Problem-specific architectures implement a model and a given topology; the weights of the network are programmable. The learning is done most of the time off-line.

(3) With algorithm-specific architectures, the model is selected a priori. Topology can be modified, and the learning is carried out by the system itself.

(4) Neural processor architectures are also called multimodel accelerators. They are much closer to a generic processor [14, 15].

(5) VLIW digital signal processors (DSP) can also be used to implement neural networks, but they are more generic processors. Many DSP chips are available, like Equator MAPCA BSP, NEC SPXK5, or Analog TigerSHARC, that include a small degree of parallelism. Some are built around a large parallel processor structure, (VLIW) linked to a scalar RISC processor, in a single core structure, such as, the Siroyan SRA328 [16], which is much more a real multiprocessor. The ChipWrights CWv8 processor core [17] is much more an SIMD processor. The RC Module NeuroMatrix NM6403 core [18], which is a real full vector/matrix parallel processor, provides scalable performances and a programmable operand width of 1 to 64 bits. This flexibility allows designers to trade precision for performance to suit their applications. The NM6403 processor includes a 32/64-bit RISC processor and a 1- to 64-bit vector coprocessor that supports vector operations with elements of variable bit lengths. The vector coprocessor, with SIMD (single-instruction multiple-data) architecture, works on packed integer-data comprising 64-bit blocks in the form of variable 1- to 64-bit words. The device is limited to vector/matrix or matrix/matrix multiplications. The vector coprocessor's core looks like an array of multipliers comprising cells that include a 1-bit memory (flip-flop) surrounded by several logical elements. Designers can combine the cells into several macrocells with two 64-bit programmable registers. These registers define the borders between rows and columns with macrocells. Each macrocell performs the multiplication on variable-input words using

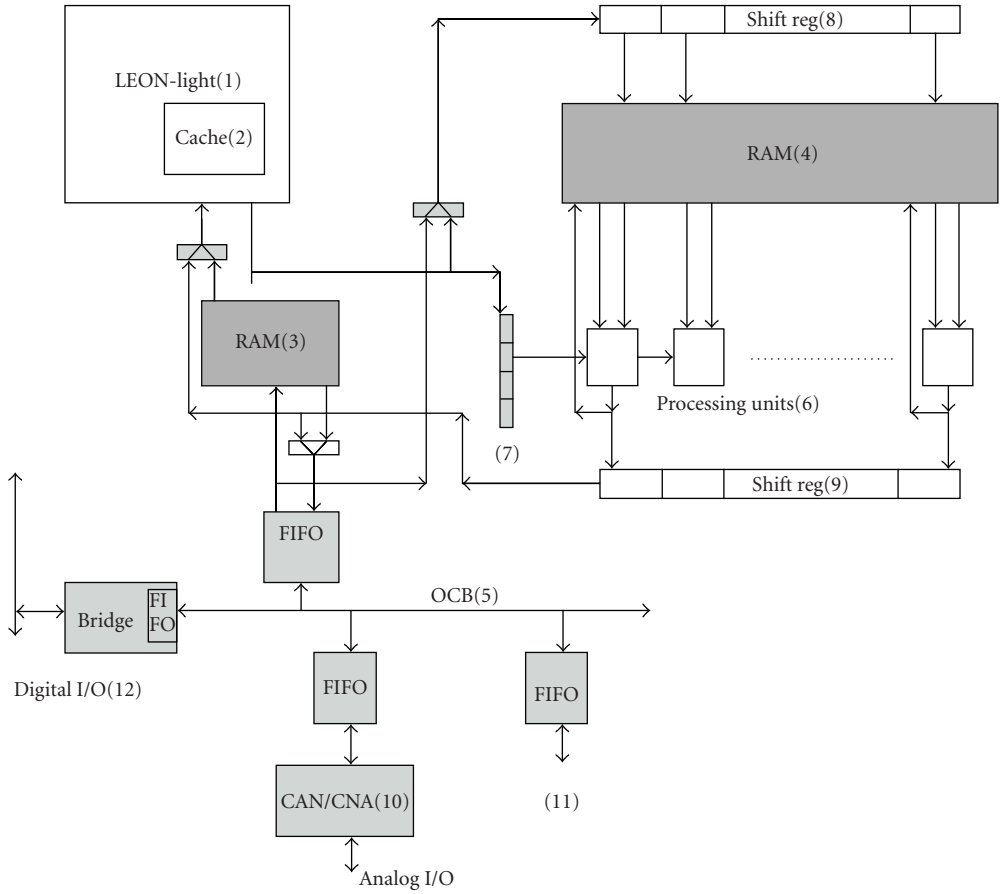


Figure 5: SoC platform architecture.

preloaded coefficients and accumulates the result from the macrocells in the column above it. The columns simultaneously calculate the results in one processor cycle. For 8-bit data and coefficients, the vector coprocessor performs 24 MAC (multiply-accumulate) operations with 21-bit results in one processor cycle. The number of MAC operations depends on the length and number of words packed into a 64-bit block. The engine's configuration can change dynamically during calculations. An application can start with maximum precision and minimum performance and dynamically increases the performance by reducing the data-word lengths.

6. THE OBJECT RECOGNITION SYSTEM

If we take the classical and simplified architecture of an object recognition system, we have the following modules: image sensor, detection, localization, and diagnosis. For our implementation, we propose a PC-based recognition system, and use a standard camera as an image sensor. Therefore, it is the detection module that we will hardware-implement using the SVM as its core. In order to be able to integrate the detection module in the PC-based system, we will use the PCI interface.

7. THE SOC PLATFORM ARCHITECTURE

A particular SoC category concerns the SoC platforms [19], an emerging technology whose main purpose is to provide a reusable silicon platform for many applications, either for several versions of a single application or even for several different applications in the same field. This is due to the growing design and fabrication costs of ASICs, which thus impose large amounts of chips. The only solution is to have more general reusable chips. The Xilinx VirtexPro II can be considered as a general-purpose SoC platform, which associates dedicated blocks such as PowerPC processors, RAM and multipliers, and a classic FPGA part that can be dynamically reconfigured.

The platform we are proposing here is dedicated to fixed-point vector/matrix operations, which are the basic operations of many signal and image processing functions. We have concentrated our effort on neural network applications.

Figure 5 depicts the general architecture of our proposed SoC platform. This platform is built around a RISC 32-bit processor linked to a parallel vector coprocessor. Both are connected to a network-on-chip (NoC) [20] that controls communications between the different parts of the system. Here the NoC is a PCI-X on-chip bus (OCB) version. We have simplified the LEON2 SPARC processor (1) in order

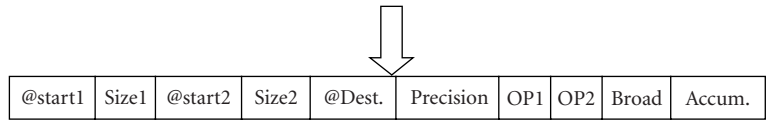


Figure 6: Configuration instruction register.

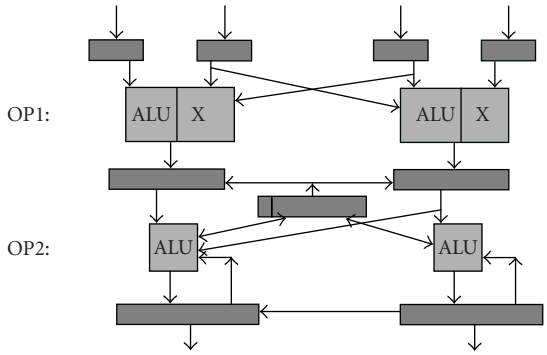


Figure 7: Multiprecision processing unit.

to communicate directly from its cache memory (2) to the dual-ported RAM (3) used to store LEON2's binary code and data. A second data RAM (4) is accessible in the memory address space of both the LEON2 processor and the external I/O subsystem (5), which is here a simple on-chip bus with its wrappers (light-gray boxes). This dual-ported RAM is the storage unit of the CP vector/matrix unit (6) which performs ALU/MAC operations loops on vector/matrix fixed-point data from the RAM, according to the instruction register (7) which provides the configuration of the processing units. This register is detailed in Figure 6. The ALU allows any kind of operations to be executed, leading to a richer instruction set than the simple MAC operations of most similar approaches such as the NeuroMatrix chip [18].

This is a double register operating in ping-pong mode. This register is reconfigured for each new matrix operation. The configuration that is provided to the vector processing unit is the size, step, and addresses of the loops, the precision of data, and the operations performed with or without accumulation.

Here it is an example of vector/matrix operation with accumulation.

The (8) and (9) registers are used to shift input and output data in and out of the vector RAM. These registers can also broadcast input and output data in the case of vector/matrix operations to be treated as matrix/matrix operations.

The multiprecision unit is presented on Figure 7. This is a version with only two different input precisions (8-bit and 16-bit) in order to simplify the presentation. The first OP1 operator is either an 8×8 multiply or an 8-bit ALU. The 16-bit result can be accumulated with the 32-bit OP2 operator. The two 8-bit multiply operators (OP1) can also be combined to perform a 16-bit multiply in two clock cycles, using the accumulation operator (OP2) to perform the two additions. A 16-bit MAC is thus performed in four clock cycles, that is, two for the four multiply operations, one for the last addition of the 16-bit multiply results, and one for the final accumulation. The accumulation is pipelined with the preceding operation, which is thus treated every clock cycle for an 8-bit MAC, every three cycles for a 16-bit MAC, and every five cycles for a 32-bit MAC, that is, every $N+2$ cycles with N being the number of bytes of data precision. The main limitation of our proposed architecture is the vector data precision which must be a multiple of 8 bits, which however is often the case in image processing. The counterpart is the lower complexity of the logic which leads to higher clock frequencies, compared to the NeuroMatrix solution which is a 1-bit multiple with a lower clock frequency.

The last part of the system is the I/O subsystem, which has to feed the processor with data. The OCB is used as the central communication subsystem between the processor/coprocessor and the external analog (10) and digital (12) ports. Other components can also be integrated on the OCB (11), like other processor/coprocessor couples, in order to build a complex system. A single large coprocessor with many processing units would be difficult to manage due to the limited available data and instruction parallelism as well as the long distances between units which would affect their communications and the clock frequency. Here is Algorithm 1 an example of configuration obtained from the original and adapted C SVM source codes of Algorithms 2 and 3. The coprocessor here executes only the two internal loops. These internal loops are packed in a function.

The Acquisition$(\cdot)$ function starts a DMA with the Co-Processor RAM, and the Sync$(\cdot)$ function waits the end of the coprocessor treatment and switches the configuration register and RAM banks to prepare the next treatment. The reconfiguration is performed by program (LEON2 C code), with dynamically constructed vector instructions (configurations). We have developed a preprocessing C parser which analyzes CP_name$(\cdot)$ functions, which have to comply with the pattern of Algorithm 4. This preprocessing links the parameters of the loops to the dedicated C library function which will dynamically, that is, at run-time, fill the fields of a configuration instruction register which will then be launched to the CP core (coprocessor). These configurations represent the nature of the processing, that is, the SVM processing. This vector coprocessor architecture is a good compromise between a fully hardwired solution and a fully programmable general-purpose solution. A first small C library has been designed for our SVM experiments. This approach can be compared to the Neuromatrix one [18], which is the only comparable product on the market. Its approach is

```
int CP_Recognition(int nb_sensors, int nb_supports)
  { int k, j;
        for(j = 0; j < nb_supports; j + +)
        for(k = 0; k < nb_sensors, k + +)
              oo[j]+ = support_vectors[nb_sensors* j + k]*sample[k];}
```

Support_vector	nb_Supports	sample	nb_sensors	oo	8	+	x	no	acc.

Algorithm 1: CP_Recognition(·) function and equivalent configuration.

```
/** RECONNAISSANCE ******************************************************/
/**********************************************************************/
void
recognition(sample, nb_samples, nb_sensors, std, support_vectors,
            support_weights, support_threshold, nb_supports, kernel,
            classe, gausse, var_thresh)
    int          *sample;          int        nb_samples;
    int          nb_sensors;       float      std;
    int          *support_vectors; float      *support_weights;
    float        support_threshold; int       nb_supports;
    char         *kernel;          int        *classe;
    double       var_thresh;
{
    double       out;
    double       oo;
    int          i, j, k;
    struct timeb debut,final;
    FILE         *fichier;
    for(i = 0; i < nb_samples; i + +){
        out = 0.0;
        for(j = 0; j < nb_supports; j + +){
            oo = 0.0;
            if (kernel[0] == 'p'){
                for(k = 0; k < nb_sensors; k + +)
                    oo + = (support_vectors [nb_sensors *j + k]* sample[nb_sensors *i + k]);
                out + = support_weights [j]* pow (oo +1.0 std);
            }
        }
        OUT + = support_threshold;
        if(out >= var_thresh)
            classe [i] = 1;
        else
            classe [i] = −1;
    }
}
```

Algorithm 2: Original C-code of the SVM generalization function. The part of the code that is executed on the coprocessor is underlined.

```
for (i = 0; i < nb_samples; i + +){out = 0;
    for (j = 0; j < nb_supports; j + +) oo [j] = 0;
    CP_Recognition(nb_sensors,nb_supports);   // non blocking
    Acquisition(sample);                      // non blocking
    Sync();                                   // blocking
    for (j = 0; j < nb_supports; j + +) out + = support_weights [j]*pow(oo[j] + 1);
    out + = support_threshold;
    if (out >= var_thread) classe [i] = 1; else classe [i] = 0; }
```

Algorithm 3: Adapted C-code loops of SVM generalization function.

```
For (J=start2; J <size2; J+ =step2) {
  For (I =start1; I <size1; I+ =step1) {
    Res[J] = Res[J] op2 (RAM1[I] op1 RAM2[I]); } }
```

Algorithm 4: General parallel loop pattern.

based on a static compile-time generation of configurations, which is most of the time sufficient and as easy to program as our solution, but dynamicity becomes more and more important. This is particularly important when the reconfiguration needs to be dependent on the previous results of a processing, in a nonpredictable way. The heart of our learning procedure is based on the classification procedure, which is evaluated here. The dynamic nature of the parameters is used here in the learning phase, which calculates the interesting support vectors, their number, and their size according to the quality of the classification. These results can be reinjected in the classifier treatment in order to size the final classifier parameters. Thus, this nonsupervised approach leads to an automatic parameterization of the classification treatment. This SVM solution, which is optimal in terms of database classification, can thus be used as an automatic solution to many treatments, which can be adapted and solved by means of classification. This kind of approach is only possible if a large-size hardware is available, that is, in a near future.

This application could be also implemented on the Neuromatrix chip, but with lower execution time (or higher costs). The scalability of the application, which is linear for matrix operations, can be dealt within two ways. First, when the size of the loops is higher than the size of the coprocessor, a second internal level of loop is performed in the coprocessor structure by means of RAM address management, that is, with a circular data mapping in the RAM array. Second, when the data size is higher than the RAM size, the treatment has to be divided in smaller parts manually at compile time, either on the same coprocessor by serialization, or on several different coprocessors linked together through the network-on-chip, which is here the PCI-X on-chip bus. A future data cache RAM array architecture is under study in order to mask this limitation to the programmer. Both solutions lead to performances or cost impact due to serialization of operations.

8. RESULTS

8.1. The choice on SVM parameters

The retained kernel function of the SVM machine is the polynomial, because of the obtained performances and also because its hardware implementation is relatively easier. The principal parameter of the polynomial kernel function is the degree of the polynomial. Although it is possible to make an implementation with a variable polynomial degree, we took as basic choice a polynomial of degree 2 (a higher degree would impose the use of wider data buses). Considering that our principal application is the matrix barcodes detection, we took 16×16 pixels as the detection window size for the hardware implementation. This size is not important to the application level. Table 4 summarizes the hardware parameters and Table 5 the behavioral specification of the SVM classifier.

Table 4: Hardware parameters summary.

Parameters		Memory size
Input dimension	256 pixels	32×32 bytes
Input precision	8 bits	—
Kernel	Polynomial	—
Degree	2–4	—
Number of SV	128	128×256 bytes
Weights precision	24 bits	128×3 bytes

Table 5: SVM core behavioral specification.

Kernel function	Polynomial
Degree of the kernel function	2/3/4
Number of support vectors	128
Input dimension	64 or 256

8.2. The choice of the hardware parameters

We have chosen to implement the SVM classifier on a SoC platform in order to exploit the parallel nature of the SVM algorithm. The active logical blocks and the interconnection buses normally consume the surface of silicon of an ASIC or FPGA circuit. For a few years, interconnection busses became the main consumers of silicon surface, due to the complexity of algorithms and circuits.

Input data. The values of pixels are coded in 256 gray levels, therefore all memories with pixel values will be used as a multiple of 8 bits. So each element of the support vectors will correspond to an 8-bit value.

Weight Data. It is the only data whose size is not defined by the specification. It is significant to define its size precisely, in order not to modify the recognition performances significantly. Although the values in the software implementation, are float values, in the hardware implementation, we use fixed point to avoid the use of the floating-point operators. The results of weights precision analysis were obtained from the same database used for testing the SVM algorithm. We vary the precision (number of bits) of the weights and we obtain the percentage of good detection and of bad classifications, the rest corresponding to false alarms.

In opposition to the results obtained by Bermak [21] and those shown in literature [22] where the average is 8 bits, the necessary precision to have the same success rate that we obtained by software is near 16 bits. It is a relatively high precision compared to the implementations shown in literature. Different factors can explain this result: for off-line learning we have in general a more significant precision [22]. Afterwards, the weights obtained during this learning process must be approximated to their hardware precision. In our case, the learning precision is maximal. Other models have

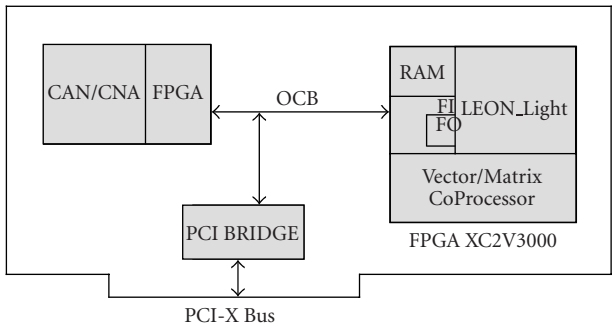


Figure 8: FPGA-based prototyping platform.

Table 6: VHDL XC2V3000 synthesis summary.

Blocks	Size	Frequency
32-bit LEON2-light	14%	50 MHz
64-bit OCB interface	6%	133 MHz
64 8-bit coprocessors	60%	166 MHz
LEON2 RAM	32 KB	—
Coproc. RAM	128 KB	—

fewer neurons than the SVM: this requires less precision for a hardware implementation. And finally, we recall that the hyperplane of separation in the case of the SVM is in a dimension which is much higher than the input dimension, and that the solution is built up using the support vectors. The 32 bits of the processor outputs are sufficient to provide the result to the generalization function, which operates on the data weights. The LEON processor performs this last function, which is limited in complexity, sequentially, and in pipeline with the matrix product.

8.3. Prototyping platform

We have designed a general rapid prototyping platform dedicated to SoC emulation. The central board connects a CAN/CNA module with a Xilinx XC2V3000 FPGA and a PCI-X controller. This general-purpose board is presented in Figure 8. A more complex system can be built with several boards on the PCI-X bus, which corresponds to the OCB of our final SoC. We have implemented and validated the presented application on this single board. The synthesis results obtained are presented on Table 6. The vector coprocessor RAM is organized in two 1 KB RAM per processing unit. The peak performances of 10 Giga MAC/s have been reached with this application. As a comparison, the number of gates of our chip is nearly the double compared to the Neuromatrix core. Also, the main vector/matrix product consists of $256 * 88$ 8-bit multiplies, that is, $256 * 88/64 = 352$ clock cycles compared to the $16 * 88 = 1408$ clock cycles with the evaluated Neuromatrix chip. We have thus obtained an efficient solution, easy to program. A large SoC will be studied on CMOS 0.13 μm technology ASIC in order to obtain real-time execution with more important applications.

9. CONCLUSION

Platform-based design (PBD) is the best-validated industrial approach for achieving high reuse in SoC design, and incurs the lowest risk in derivative creation via user programmability. Although these platforms already exist in some application domains, their design process is largely ad hoc. Furthermore, despite high development costs, such platforms tend to be difficult to program, and very little software support is available. Our proposition attempts to fill this gap. Our approach is to provide a general-purpose neural network application customized by a learning phase instead of explicit programming which avoid tedious designing effort. Such a solution is only possible with large hardware platforms. We have proposed in this paper a sizeable SoC platform dedicated to regular image and signal processing involving matrix operations. We have illustrated its implementation capabilities with the SVM neural network application, which performs object recognition of any kind (image or signal). A user-friendly interface is under construction. Also a future ASIC SoC implementation will demonstrate the feasibility of our approach on realistic objects recognition. With such a system, it is possible to obtain an automatic object recognition/classification based on a learning phase, which automatically configures the recognition engine, and then obtain a real-time toolbox for any object classification.

ACKNOWLEDGMENTS

We thank Intermec for the image databases used to show the performances of the SVM method for a real application. Roberto Reyna-Rojas thanks the Mexican Research Council for its financial support through the Studentship 111030. This work was supported in part by the Mexican Research Council CONACYT under Student Scholarship 111030.

REFERENCES

[1] C. M. Bishop, *Neural Networks for Pattern Recognition*, Oxford University Press, Oxford, UK, 1995.

[2] V. N. Vapnik, *Statistical Learning Theory*, Wiley-Interscience Publication, New York, NY, USA, 1998.

[3] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proc. 5th ACM Workshop on Computational Learning Theory (COLT '92)*, pp. 144–152, Pittsburgh, Pa, USA, July 1992.

[4] C. Burges, *A Tutorial on Support Vector Machines for Pattern Recognition*, Kluwer Academic Publishers, Boston, Mass, USA, 1998.

[5] E. Osuna and F. Girosi, "Reducing the run-time complexity of support vector machines," in *Proc. International Conference on Pattern Recognition (ICPR '98)*, Brisbane, Australia, August 1998.

[6] Y. LeCun, L. D. Jackel, L. Bottou, et al., "Comparison of learning algorithms for handwritten digit recognition," in *Proc. International Conference on Artificial Neural Networks (ICANN '95)*, F. Fogelman-Soulié and P. Gallinari, Eds., vol. 2, pp. 53–60, Paris, France, October 1995.

[7] B. Schölkopf, *Support vector learning*, Ph.D. dissertation, Technical University of Berlin, Berlin, Germany, 1997.

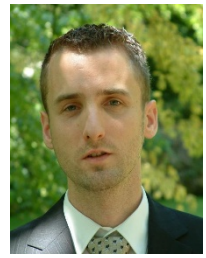
[8] V. L. Brailovsky, O. Barzilay, and R. Shahave, "On global, local, mixed and neighborhood kernels for support vector

machines," *Pattern Recognition Letters*, vol. 20, no. 11-13, pp. 1183–1190, 1999.
[9] B. Marcel, *Study on the automatic reading of two-dimensional codes by image processing*, Ph.D. dissertation, Institut National Polytechnique de Toulouse, Toulouse, France, 1997.
[10] R. Reyna-Rojas, N. Hernandez, D. Esteve, and M. Cattoen, "Segmenting images with support vector machines," in *Proc. International Conference on Image Processing (ICIP '00)*, vol. 1, pp. 820–823, Vancouver, BC, Canada, September 2000.
[11] A. Jain, R. Duin, and J. Mao, "Statistical pattern recognition: a review," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. 22, no. 1, pp. 4–37, 2000.
[12] T. Delbrück and C. A. Mead, "Analog VLSI phototransduction by continuous-time, adaptive, logarithmic photoreceptor circuits," in *Vision Chips: Implementing Vision Algorithms with Analog VLSI Circuits*, C. Koch and H. Li, Eds., pp. 139–161, IEEE Computer Society Press, Los Alamitos, Calif, USA, 1994.
[13] L. O. Chua and L. Yang, "Cellular neural networks: applications," *IEEE Trans. Circuits Syst.*, vol. 35, no. 10, pp. 1273–1290, 1988.
[14] M. F. Emirian, *Study and design of a parallel machine multimodels for the neural networks*, Ph.D. thesis, Institut National Polytechnique de Toulouse, Toulouse, France, 1996.
[15] M. Viredaz, *Design and analysis of a systolic array for neural computation*, Ph.D. thesis, Ecole Polytechnique Fédérale de Lausanne, Lausanne, Switzerland, 1994.
[16] P. Clarke, "Siroyan implements clustered DSP architecture," *EE Times*, October 2001.
[17] M. Long, "ChipWrights Intros New Visual Signal Processor," *ECN magazine*, October 2003.
[18] VLIW/SIMD NeuroMatrix Core, Research Center MODULE, Moscow, Russia.
[19] A. Sangiovanni-Vincentelli and G. Martin, "Platform-based design and software design methodology for embedded systems," *IEEE Des. Test. Comput.*, vol. 18, no. 6, pp. 23–33, 2001.
[20] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," *IEEE Computer*, vol. 35, no. 1, pp. 70–78, 2002.
[21] A. Bermak, *Sysneuro: un circuit systolique neuronal multiprécision pour des tâches de classification*, Ph.D. thesis, Université Paul Sabatier de Toulouse, Toulouse, France, 1998.
[22] P. Moerland and E. Fiesler, "Neural network adaptations to hardware implementations," in *Handbook of Neural Computation*, chapter E1.2, pp. 1–13, Institute of Physics Publishing, Bristol, UK; Oxford University Press, New York, NY, USA, January 1997, IDIAP Research Report IDIAP-RR 97-17.

Roberto Reyna-Rojas was born in Tehuantepec, Mexico, in 1972. He received the Electronics Engineering degree from the Universidad Autónoma Metropolitana, Mexico City, in 1994, and the Computer Science Engineering degree from ENSEEIHT, Toulouse, France, in 1997. He obtained the Ph.D. degree from the National Institute of Applied Sciences, Toulouse, France, in 2002. He is actually working for the French Space Agency CNES in VLSI circuits failure analysis, and particularly failures provoked by ESD events. His research interests are in the areas of VLSI design and reliability, development of integrated neural networks, and specific signal processing architectures.

Dominique Houzet received the M.S. degree in computer sciences in 1989 from Paul Sabatier University, Toulouse, France, and the Ph.D. degree and HDR degree in computer architecture in 1992 and 1999 both from INPT, ENSEEIHT, Toulouse, France. He worked at IRIT Laboratory and ENSEEIHT Engineering School from 1992 to 2002 as an Assistant Professor and also as a Digital Design Consultant with SME. He is an Associate Professor in the Department of Telecom, the National Institute of Applied Sciences, and IETR Laboratory, Rennes, France, since 2002. He has published a number of research papers in the area of parallel computer architecture and SoC design and a book on VHDL principles. His research interests include codesign and SoC design methodologies applied to image processing and radiocommunications. He is a Member of the IEEE Computer Society.

Daniela Dragomirescu was born in Bucharest, Romania, in 1972. She received the Electronics Engineering degree from Bucharest Polytechnic University, Romania, in 1996, and her Ph.D. degree from the National Institute of Applied Sciences, Toulouse, France. She has been an Associate Professor at Toulouse National Institute of Applied Sciences (INSA) since October 2001 and a researcher at Laboratory for Analysis and Architecture of Systems (LAAS) French National Center for Scientific Research (CNRS). Her research interests are in the area of systems-on-chip and hardware architectures for signal processing.



Florent Carlier received the Ph.D. degree from the National Institute of Applied Sciences, Rennes, France, in 2003, where he specialized in neural networks for communication systems. He has been an IEEE Student Member. He began an academic career in 2004, as a Lecturer in numerical electronics and member of the Department of Informatics (LIUM), the University of Maine, Le Mans, France. His research interests include computer architecture, digital systems, protocol design, and computer programming.

Salim Ouadjaout received an M.S. degree in computer science from the National Institute of Computers (INI), Algeria, in 2000, and an M.S. degree from INP, ENSEEIHT, Toulouse, France, in 2001. He is a Ph.D. candidate in electrical and computer engineering at the Institute of Electronics and Telecommunication, Rennes, France. He is also working as a research engineer at M3Systems Inc. He has been an ACM Student Member. His research interests include design methodology, interface synthesis, and micronetworks for SoC.