Buletinul Ştiinţific al Universităţii "Politehnica" din Timişoara

Seria ELECTRONICĂ şi TELECOMUNICAŢII
TRANSACTIONS on ELECTRONICS and COMMUNICATIONS

Tom 51(65), Fascicola 1, 2006

SVM Classifier using LUT-based RAM on a Spartan 3 FPGA

Albert A. Fazakas[1], Mihaela Cârlugea, Lelia Feştilă

Abstract – Support Vector Machines are widely used in pattern recognition, being the newest achievements in neural network structures. This paper presents an implementation example of an SVM classification function using a Spartan3 FPGA device. A Block Ram based implementation is compared versus a distributed LUT-based RAM one. Aspects regarding memory geometry and instantiation are presented. The number of required clock periods and the maximum clock frequency is calculated and a speed comparison of the implemented system with software running on a PC targeting the same application is also made

Keywords: SVM, Block RAM, LUT-based RAM, FPGA

I. INTRODUCTION

Support Vector Machines (SVM) are considered to be the newest achievements on neural network structures [1]. In Chapter 2, the basic idea of the SVM is presented and their advantages are highlighted. Chapter 3 presents an application example for classification of the Ibermatica database images. The role of the classification function is to decide whether a particular image is face- or non-face image. The major advantage offered by an FPGA device, the possibility to implement parallel structures is used in the system implementation. An implementation example of the kernel function using Block RAM components follows. Chapter IV presents the implemented system that is based on distributed RAM cells also called LUT (LookUp Table) RAM cells, to tackle the drawbacks that come with the Block RAM based implementation. Approximations made in order to reduce the data bus widths are also presented. The total number of clock cycles required for the classification function to perform its operations is calculated and the maximum clock frequency is determined. Finally, a comparison between the speed of the system implemented on the FPGA and the speed of a classification software running on a PC is also presented.

II. SUPPORT VECTOR MACHINES

In the nineties, the neural networks knew a very significant importance in the scientific and engineering domains. Industrial products are offered today on the market with a real success even if we do not have the associated physical model for diagnosis. It is necessary to consider the neural networks as a manner of 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. The model is the most recent proposition on neural network structures [1]. This model is founded on the statistical learning Theory. The Support Vector Machine model consists of 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 SVM make a distinction of two classes by finding a decision surface constructed from certain points of the entire learning database, called Support Vectors A second important idea of Support Vector Machines is the use of kernel functions. The kernel functions were proposed to be able to build nonlinear algorithms from linear algorithms by calculating the inner product not in the input space but in the feature space. By using kernels it can be taken into account the statistics of greater order without a combinatorial explosion of the complexity than it would have met even for moderate values of examples and the dimension of the kernel function. The most used kernel functions are the polynomial, sigmoid (neural network) and the Radial basis function.

II.1. The Support Vectors

Vapnik [2]. proposes a representation of a 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 *xi* for which equality $y_i((w_0x_i+b_0)=1$ holds. Concretely, they are the closest points to the optimal

[1] Facultatea de Electronică, Telecomunicaţii şi Tehnologia Informaţiei, Catedra de Bazele Electronicii, str. Gh. Bariţiu Nr. 26-28, 400027 Cluj-Napoca, e-mail Albert.Fazakas@bel.utcluj.ro

hyper plane. For all the other examples, there is thus a factor α=0 that eliminates them from the solution. We thus know that the decision function is calculated from the examples that are on the margin, presented in figure 1. In the non-linear case, it is enough to replace the scalar products *(x × x_i)* by kernels *k(x, x_i)*.

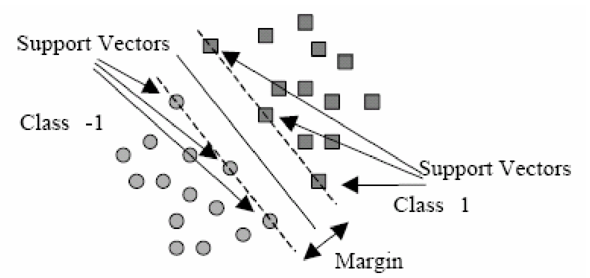

Fig. 1. The support vectors are the closest vectors to the optimal hyperplane [1].

II.1. The Polynomial Kernel

There are three options for the selection of the kernel function of the SVM method: Polynomial, RBF or sigmoid Neural networks [1]. The Sigmoid Neural network kernel function option was rejected because of the difficulty of a possible hardware implementation. Moreover in the literature the performances obtained with this kernel function were lower than those obtained with the two others.
The following is the general equation of the SVM decision function for classification:

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

Where:
y_i $a_i = w_i$, are the networks weights,
X_i, are the support vectors of the solution,
b, is the threshold of the function, and
$K(X,X_i)$.is the kernel function.
As it can be seen, the solution is the sign of the addition, so this is the generalization function for two-class's classification. In our case, the kernel function is then the polynomial function of degree d:

$$K(X, X_i) = (X^T \cdot X_i + c)^d \quad (2)$$

III. IMPLEMENTATION EXAMPLE

III.1. The classification function parameters

Our objective is to implement a classification function for the image database provided by Ibermatica [3]. The database is composed of 8X10 pixel resolution 8-bit grayscale images. The classification problem for the SVM is to decide whether or not the image is representing a human face. Figure 2. a) shows a positive example and figure 2. b) a negative example from the Ibermatica database training set.

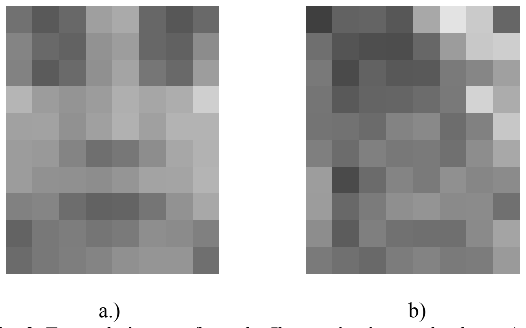

Fig. 2. Example images from the Ibermatica image database a) positive (face image) example b) negative (non-face image) example

Training was done for the SVM on 112 images from the image set. The SVM parameters were:

- Degree of the kernel function d=2;
- Constant c=1
- Threshold value b=1.5215772
- Number of support vectors: 17
- Feature index i.e. image size: 80

The implementation is done on a SPARTAN 3 XC3S200 device, due to its availability on the Digilent S3 development boards.

III.2. Spartan-3 Block RAM implementation considerations

According to the specifications above, the data size required to store the image to be classified and the support vectors is

$$17 \times 80 \times 8 + 80 \times 8 = 11520 \text{ bits} = 11.25 \text{ Kbits} \quad (3)$$

This amount of data can be easily fit into the considered device [4]. Moreover, the Spartan-3 device features twelve 18-Kbit dual-port RAM memories, also called Block RAM (BRAM) memories. Results that a single BRAM is sufficient to fit the amount of data considered. However, in this case the whole data is processed sequentially and the number of clock cycles required to implement one kernel function is 80. This is multiplied by the number of support vectors and some extra clocks are added for pipelining purposes.
The required number of clock cycles can be significantly reduced if the advantage of an FPGA implementation, the possibility of parallel processing is used wherever is possible.
Basically, the classification function operations consist in a set of multiplication and summing-accumulate, i.e. MAC operations. The embedded 18-bit multipliers will be used for implementing the kernel function. For the weighting operation, a multiplier larger than of 18 bits will be needed due to the increase of the data bus width as result of the multiplication-summing procedure in the kernel function. A number of 10 multipliers will be considered to work in parallel for implementing the kernel function. The number of ten was chosen because the image size considered is dividable with

10, in fact, most of the image sizes feature this property, making the application easily adaptable to different image sizes.
Therefore the values of the X image and the X_i support vectors are distributed in ten RAM blocks; each block basically stores eight pixels of the X image and the corresponding 17X8=136 pixels of the X_i support vectors. The RAM blocks can be configured into various geometries [5]. For the specified application, the geometry chosen for the BRAm-s is of 1KX16 bits. The X pixels are stored in the upper and the X_i pixels in the lower byte of the memory. Basically only 136 locations are used in each memory block from the available 1024. Assuming a number of 17 support vectors, the system configured in this way supports an image size of 600 pixels, with any aspect ratio.
Fig. 3 shows the block schematic of the kernel function implemented with Block RAM components and Table 1 shows an example for the placement of the X() image data and the X_i() support vector data in the BRAM-s, i.e. a memory map example. Obviously, the data can be placed in various ways into the memory until the placement is uniformly distributed and the stored X() image data corresponds with the X_i() support vector data stored at the same locations.
Each block memory contains eight pixels of the analyzed image, repeated the number of support vector times. Due to this redundancy the memory map is inefficiently organized, however this placement of the data insures one multiplication at each clock cycle, therefore the kernel function is able to perform its operations for one support vector in a total number of 8 clock cycles.

Table 1. BRAM memory map example

Addr.	BRAM0		BRAM1			BRAM9	
	[15:8]	[7:0]	[15:8]	[7:0]		[15:8]	[7:0]
000	X(0)	X_1(0)	X(8)	X_1(8)	.	X(72)	X_1(72)
001	X(1)	X_1(1)	X(9)	X_1(9)	.	X(73)	X_1(73)
.	.	.	.	.	.	.	.
007	X(7)	X_1(7)	X(15)	X_1(15)	.	X(79)	X_1(79)
008	X(0)	X_2(0)	X(8)	X_2(8)	.	X(72)	X_2(72)
009	X(1)	X_2(1)	X(9)	X_2(9)	.	X(73)	X_2(73)
.	.	.	.	.	.	.	.
00F	X(7)	X_2(7)	X(15)	X_2(15)	.	X(79)	X_2(79)
.	.	.	.	.	.	.	.
086	X(6)	X_{17}(6)	X(14)	X_{17}(14)	.	X(78)	X_{17}(78)
087	X(7)	X_{17}(7)	X(15)	X_{17}(15)	.	X(79)	X_{17}(79)
088	0	0	0	0	.	0	0
.	.	.	.	.	.	.	.

The summing and accumulating circuit performs the unsigned sum of ten 8-bit numbers. Two-input 8-bit adders were used on more levels to add all the ten numbers, therefore the required number of the adders is 5 on the first level, 2 on the second level and 1 on the third and fourth levels. The multiplier outputs and each summing level outputs are registered for pipeline considerations. Results that the total number of required clock cycles to perform the kernel function operations, for all of the support vectors increases with 6, becoming

$$17 \times 8 + 6 = 142 \text{ clock cycles} \tag{4}$$

Because the image data has to be repeated in the BRAM blocks, results that loading a new image into the memory implies sweeping the whole used memory

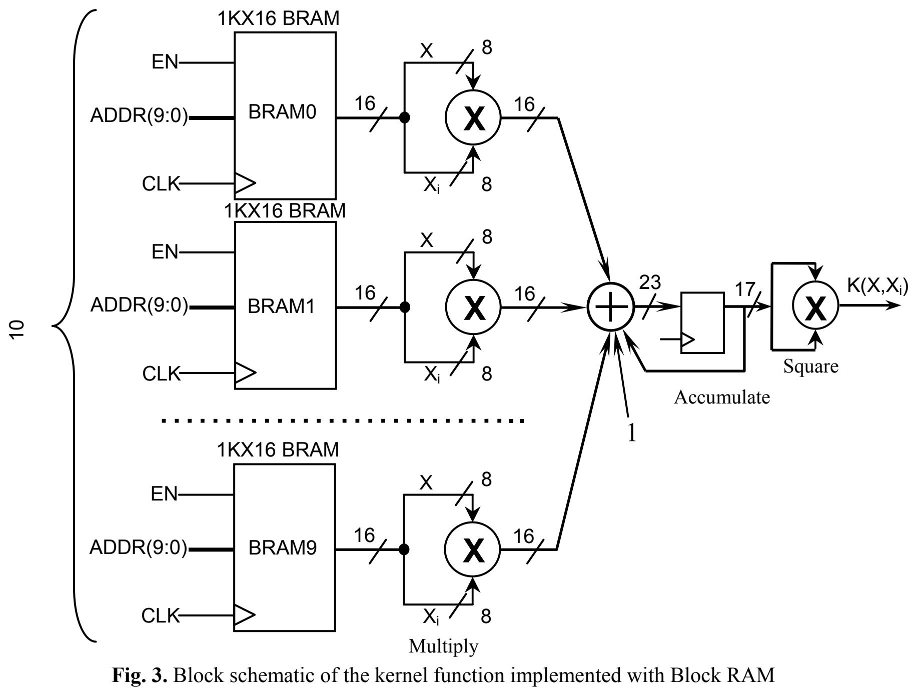

Fig. 3. Block schematic of the kernel function implemented with Block RAM

address space, i.e. loading a new image would take a minimum of 17X8=136 clock cycles, assuming that the ten RAM blocks are loaded simultaneously.

Loading the BRAM memories with new data is eased by the fact that the Xilinx Block RAM components are true dual-port memories, allowing simultaneous read and write from two different ports. The only restriction is applied to the fact that the same memory location cannot be accessed simultaneously from both ports. The dual-port RAM facility allows loading the new image on the second access port while the first one is used for the kernel function operations. In order to avoid address conflict, data loading starts with one clock cycle earlier i.e. memory write is performed on the current memory address-1. Taking into account that the support vector data i.e. the lower byte in each memory is not changing when a new image is loaded, the lower byte is buffered and reloaded in the memory with the new image data. The incoming image data is also buffered and formatted in the remaining two BRAM components. The image data loading system is not shown in fig. 3. due to the lack of space.

Taking into account that the kernel function multiplies and accumulates unsigned data, the maximum possible result from the MAC operations of the kernel function i.e. the maximum number at the output of the accumulator can be

$$255 \cdot 255 \cdot 10 \cdot 8 + 1 = 5{,}202{,}001 \qquad (5)$$

The number above can be represented on 23 bits. However, only the most significant 17 bits will be taken into account as the result of the kernel function, to be able to use the remaining embedded multiplier that accepts up to 18-bit signed or 17-bit unsigned operands. It means that the kernel function is scaled with

$$K(X, X_i) = \left(\frac{X^T \cdot X_i + 1}{2^5}\right)^2 = \frac{\left(X^T \cdot X_i + 1\right)^2}{1024} \qquad (6)$$

Other scaling operations will result from the weighting and summing operations that follow the kernel function in the classifier implementation.

IV. LUT-BASED RAM IMPLEMENTATION

IV.1. Distributed memory considerations

Although the BRAM-based implementation offers operation at a high-speed by reducing the number of the clock cycles required to calculate the kernel function result, it suffers from significant drawbacks.

First, the incoming image data has to be buffered and formatted to be distributed across the BRAM components. The data formatting and RAM loading circuit takes significant resources from the FPGA. Second, the SVM classifier based on BRAM-s cannot be implemented in an embedded system together with the Xilinx proprietary MicroBlaze or PowerPc soft processor systems, because these systems use primary the BRAM-s for processor data and code memory purposes, making these components partially or completely unavailable for custom design.

In order to overcome to the incoming data formatting and distributing problem, a system with distributed memory cells was considered, that can be implemented by the Xilinx Distributed RAM (also called LUT RAM) feature

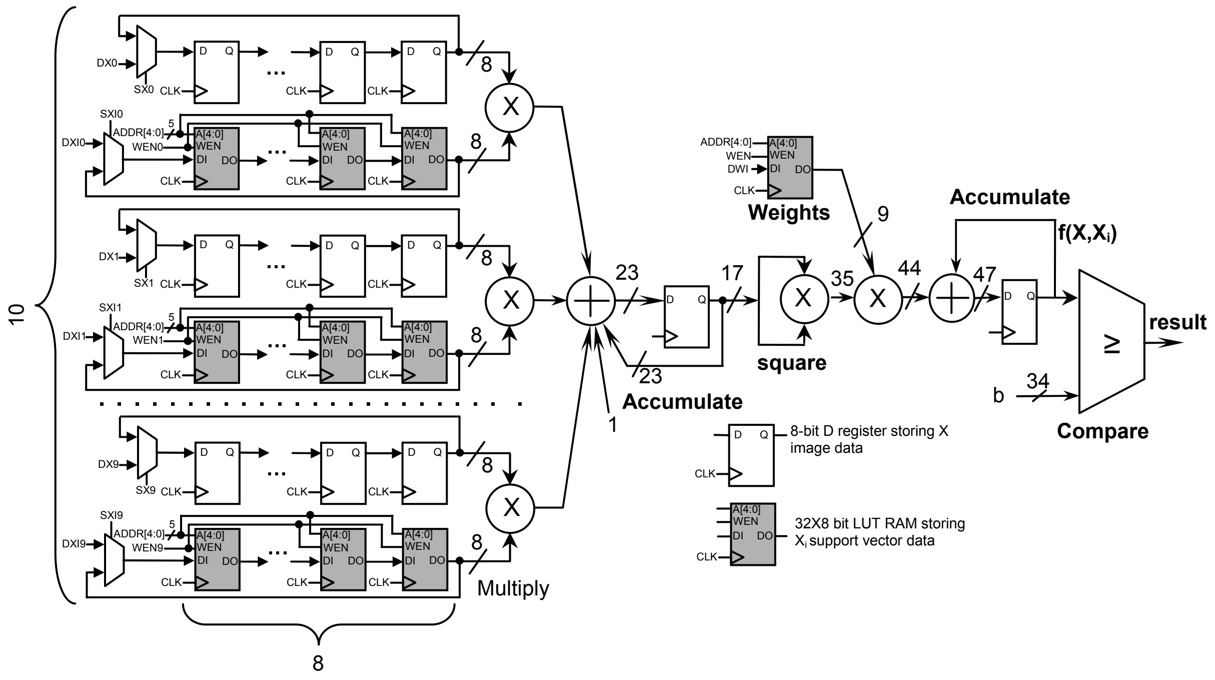

Fig. 4. Block schematic of the SVM classification function implemented with LUT RAM blocks

The LUT memory cells are built on the Configurable Logic Block (CLB) logic function generator circuits, also called Lookup Tables. In the Spartan 3 FPGA structure, one CLB consists of four slices from which two of the slices contain two 4-bit lookup tables that can be configured as dual-port or single-port RAM memories. Therefore one CLB can have up to 32 bits of dual-port memory or 64 bits of single-port memory. Results that one 32X1 bit size distributed RAM fits in a single CLB slice.
Figure 4 presents the block schematic of the implemented SVM classifier function. The system is an improved version of the SVM classifier presented in [7], where D registers were considered to store both the image and support vector data. Due to the fact that in that case the number of the D registers used is too large, taking most of the FPGA D flip-flops, the support vector data is stored in the LUT RAM blocks while the image data is stored in the adjacent D flip-flops.
The left part of Fig. 4 until the squaring circuit represents the implementation of the kernel function. The white cells are representing the D registers storing the X image data, while the gray cells the LUT RAM blocks storing the X_i support vectors. The implementation is similar to the one presented in Fig. 3 regarding the number of parallel multiplications and the way the first accumulator circuit is organized. Both the X input vector and the X_i support vector memories are organized in a circular FIFO memory matrix with a size of 10X8, resulting a number of 80X8 cells placed in the FPGA.
To store the X_i data, the distributed memory cells are organized in 32X8 bit size, allowing storage of up to 32 support vectors. A specific LUT RAM cell stores the data from the support vectors according to the expression

$$\mathrm{MEM_I}\left(\mathrm{ADDR}\right)=\mathrm{X_{ADDR[4:0]}}\left(\mathrm{I}\right) \tag{7}$$

where I is the index of the memory cell and ADDR represents the 5 bit address of the memory. Therefore the addresses of all the memory cells are connected together, creating a multiple page memory, each page containing one support vector.
Due to the circular structure of the X and X_i memories, at every 8 clock periods the memory content will be reloaded to its initial value and the address is incremented. Correspondingly, the output summing and accumulating circuit advances with one clock.
The SX0...SX9 and SXI0…SXI9 selector signals allow loading of new image and support vector data through the DX0...DX9 and the DXI0…DXI9 data lines, respectively. Formatting and loading a new image data is significantly simpler comparing to the BRAM implementation. For example, the DX0…DX9 data lines can be connected together and the SX0…9 selector lines decide which row will be loaded with the new image data. In the case of using a high-speed I/O peripheral, data can be loaded in parallel in the ten rows when the last support vector product is calculated. In the same way, a new support vector data can be loaded into the distributed RAM cells through the DXI0...9 data lines, controlled by WEN0...9 write enable lines or from the SXI0…9 selector lines.
The ADDR lines also select the corresponding weight of the support vector from the weight memory, built as well on LUT RAM cells, organized in a 32X9 bit format, with the following considerations: Initially the weights were in floating point format. Due to the fact that floating point operations would require too wide data buses and considerably more computing time, obviously were be transformed into fixed point format. The smallest weight of the support vectors for the application considered in this paper has the absolute value of

$$y_{16}\cdot\alpha_{16}=9.7740856\text{e-}014 \tag{8}$$

Note that only several decimal places were shown. The highest value of the weights, in absolute value, is

$$y_7\cdot\alpha_7=2.006978\text{e-}011=205.3\cdot y_{16}\cdot\alpha_{16} \tag{9}$$

If $y_{16}\cdot\alpha_{16}$ is normalized i.e. scaled to 1, then all of the weights can be represented on 9 bits with sign, meaning that all of the weights will be scaled with $\dfrac{1}{y_{16}\cdot\alpha_{16}}$. In order to avoid an unattended change of the classification function sign from equation (1), results that the threshold *b* has to be also scaled. Taking into account from relation (6) that the kernel function is also scaled, equation (1) will become

$$\mathrm{f}(\mathrm{x},\alpha)=\mathrm{sign}\left(\sum_{i=0}^{16}\mathrm{y_i}\alpha_\mathrm{i}\mathrm{K}\left(\mathrm{X_i},\mathrm{X}\right)+b\right)\frac{1.023\text{e+}13}{1024} \tag{10}$$

Thus, the *b* threshold becomes 1.52026e+10. This value can be represented on 34 bits, from there results the bus width for the threshold *b*.
The weighting multiplier has the size of 35X9 bits, where the 9 bits are signed, being wider than the 18 bit embedded multipliers, resulting that it has to be implemented by the FPGA lookup table logic.

IV.1. Distributed memory implementation and speed considerations

Xilinx ISE provides behavioral VHDL templates that can be used to instruct the synthesis tool to extract block RAM or distributed RAM from the code. Although the block schematic in Fig. 4 can be easily described in behavioral VHDL, it was found that by describing the circular multiple-page memory from Fig. 4 in behavioral, the XST (Xilinx Synthesis Tool) will extract D flip-flops for the memory cells together with decoding logic rather than distributed RAM cells. This behavior occurs due to the fact that the synthesizer will understand to extract shift registers on 8 bits that are basically implemented with flip-flops.

Results that for implementing the kernel memory part of the system in Fig. 4., a structural VHDL or schematic approach has to be used. The distributed RAM cells were built on RAM32X8S components in the Xilinx ISE software. The RAM32X8S represents a 32X8 bit single-port distributed RAM. The X memory cells were built on FD8CER components, representing an 8-bit D register with count-enable and asynchronous clear ports. The memory loading and controlling state-machine and the multiply-accumulate circuit were described in behavioral VHDL.

Due to the similarity of the structures between the BRAM and LUT-RAM implementations, the number of clock periods required for all of the kernel function operations is the same, i.e. 142. Four extra clock periods are needed for the squaring and weighting-accumulating operations. Results a total number of 146 clock periods for the classification function operations.

The embedded multipliers are placed close to the BRAM cells in the FPGA structure, resulting lower propagation times for the BRAM implementation than for the LUT-RAM one. However, the highest clock frequency is rather limited by the propagation times of the combinational arithmetic circuits, i.e. the embedded multipliers and the unsigned adders.

The synthesis tool reported a maximum propagation time of 4.828 ns, meaning a maximum clock frequency of 207.1 MHz. For safety purposes, half of this frequency was used. Note that the squaring and the weighting multipliers, and the final accumulator have to operate at only every eight kernel memory cock periods, allowing operations at lower frequency. A quarter of the kernel memory frequency was chosen, i.e. 25MHz. In this case, without taking into account the time needed for I/O operation, i.e. data download and memory load, the total time will be

$$T = 142 \cdot \frac{1}{100MHz} + 4 \cdot \frac{1}{25MHz} = 1.58\mu S \quad (11)$$

Due to the introduction of pipelining in the kernel function operations, the time was improved versus [7]. Comparing to the expression (11), the time spend for classification of 442 images on a Pentium IV, 1.2 GHZ PC, without taking into account the time for I/O operations, was 0.02 seconds, resulting 45.2µS per image. Obviously, for larger images the time spent for the classification function implemented on the FPGA board increases due to the window sweeping technique that has to be used.

Data downloading was made through the serial interface of the S3 board. An USB connection with the PC is under development.

V. CONCLUSIONS

In this paper an example of implementation for an SVM classification function on a Spartan 3 XS3S200 FPGA device was presented. The example is made for the Ibermatica 8X10 pixel face image database. The proposed system takes the advantage of parallel circuits offered by an FPGA implementation.

Two approaches were considered to implement the kernel function, the first one based on Block RAM memory components included in the Xilinx Spartan devices, and the second one based on distributed i.e. LUT RAM memory cells. Although the Block RAM implementation is more compact and simpler, the memory content loading can represent a difficult operation. On the other hand, the BRAM based implementation cannot be made if the SVM classifier is intended to be included in an embedded system with MicroBlaze or PowerPc soft processors that use the Block RAM as primary processor memory. The solution to these problems is offered by the distributed RAM implementation, where data loading can be easily done.

Using the parallel data processing feature offered by the FPGA devices, the required number of clock periods reduces drastically. To keep the maximum clock frequency at a high level, pipelining was used for the arithmetic operation implementations. The implemented system was performing the operations more than ten times faster than a similar software application running on a PC, with the same SVM and the same images.

In the case of larger images, a window sweeping technique has to be used that increases the computation time. Using a higher density FPGA device, the number of parallel circuits can be further increased to compensate the increase of the execution time.

REFERENCES

[1]. A. Reyna-Rojas, D. Dragomirescu, D. Houzet, D. Esteve, "Implementation of the SVM Generalization Function on FPGA", International Signal Processing Conference (ISPC), Dallas (US), March 2003

[2]. Vapnik V. "Statistical. Learning Theory" A Wiley-Interscience Publication, pp. 421, 1998.

[3]. Ibermatica S.A web site,. http://www.ibermatica.es/ibermatica

[4] Xilinx inc, "Spartan-3 FPGA family; complete datasheet", DS099 January 17, 2005

[5]. Xilinx, inc, "Using Block RAM in Spartan-3 Generation FPGAs", *Xilinx Application Bulettin XAPP463 (v2.0)* March 1, 2005

[6]. Xilinx, inc, Using Look-Up Tables as Distributed RAM in Spartan-3 Generation FPGAs", *Xilinx Application Bulettin XAPP464 (v2.0)* March 1, 2005

[7]. Albert Fazakas, Mihaela Gordan, Lelia Feştilă, Laura Kovacs, "Considerations Regarding Implementation of SVM Classification Functions on FPGA", *SIITME 2005 International Symposium for Design and Technology of Electronic Packaging,* 22-25 September 2005, Cluj-Napoca, Romania, ISBN 973-713-063-4, pp. 164-167

[8]. Xilinx, Inc. "Using Embedded Multipliers in Spartan-3 FPGAs", *Xilinx Application Bulettin,* XAPP467 (v1.1) May 13, 2003