An FPGA Implementation of Linear Kernel Support Vector Machines

Omar Piña-Ramírez, Raquel Valdés-Cristerna and Oscar Yáñez-Suárez
Universidad Autónoma Metropolitana - Iztapalapa
Neuroimaging Laboratory
09370 Mexico City, MEXICO
santusay@gmail.com

Abstract

This paper describes preliminary performance results of a reconfigurable hardware implementation of a support vector machine classifier, aimed at brain-computer interface applications, which require real-time decision making in a portable device. The main constraint of the design was that it could perform a classification decision within the time span of an evoked potential recording epoch of 300 ms, which was readily achieved for moderate-sized support vector sets. Regardless of its fixed-point implementation, the FPGA-based model achieves equivalent classification accuracies to those of its software-based, floating-point counterparts.

1. Introduction

Support vector machine (SVM) classifiers were introduced in the early eighties and ever since then they have enjoyed growing acceptance for application in all sorts of classification tasks due to their reliability, generalization capacity, and proved performance superiority over other learning classification models such as multi-layer feedforward neural networks [4]. Given that the design of an SVM classifier is data driven and involves a quadratic optimization procedure, the computational requirements during SVM training are very large. Moreover, practical application of such classification engines is also limited by the vast requirements of computational power and space implied by the trained model characteristics. It has been recognized that useful online application of an SVM requires fast hardware support, both for the training and testing phases. However depending on the application (for example those in which retraining is not a requirement) SVM training can be carried out offline in a general-purpose computer, and in such cases the relevant hardware design should aim exclusively to fast evaluation of the classifiers' decision function.

Different efforts have been made towards the design of efficient architectures for SVM training and evaluation. Probably the first account of a hardware design for an SVM system is found in the work by Anguita *et al* [2], in which the authors showed an efficient implementation of a kernel based perceptron in fixed-point digital hardware. They compared the performance of the proposed hardware implementation with the results of a simulated SVM with both fixed and floating point underlying math. They obtained similar performance metrics among the different models, and considered the hardware design a better alternative due to the small amount of bits utilized for coding and computation. In a follow-up paper [3] the same authors have proposed a digital architecture for nonlinear SVM learning. Its implementation on a field programmable gate array (FPGA) was discussed, along with the evaluation of the quantization effects on the training and testing errors. The authors established that a minimum of 20 bits for individual coefficient coding was required to maintain adequate performance.

A different approach has been proposed by Khan *et al* [5], who presented an implementation of a digital SVM linear classifier using logarithmic number systems (LNS). They used LNS in order to transform the multiplication operations involved in evaluating the decision function into addition computations, which are a less consuming task. Their design was implemented into a Xilinx FPGA Spartan3 XL3S50pq208-5 device. Their analysis showed that the LNS hardware implementation had a classification accuracy equivalent to a LNS software simulation and to a floating point software implementation. Its performance was equivalent to a 20-bit fixed-point implementation, as was reported by Anguita, but the LNS version required 25% less slices of the Xilinx FPGA. Recent studies with different data sets have allowed these authors to conclude that even a 10-bit LNS architecture was guaranteed to match the performance of a double-precision floating point alternative [6].

In this paper, we focus specifically on a design aimed at fast evaluation of the decision function of a linear SVM, avoiding the complexities associated with hardware-based training. A 16-bit implementation on a Virtex II FPGA is described, which concurrently utilizes several core multipliers to accelerate calculations, and which is suitable for pipelining. Hardware description was realized using the Handel-C modelling extensions.

For BCI application, a portable implementation of data processing with low power consumption, easily scalable, and parallelism capability is required. The main constraint of the design was that it could perform a classification decision within the time span of an evoked potential recording epoch of 300 ms, which was readily achieved for moderate-sized support vector sets.

1.1. Support Vector Machines

SVMs are a class of supervised classifier models aimed at separating feature vectors **x** from two different classes with a maximum-margin hyperplane. Such a hyperplane is constructed (by any suitable SVM training algorithm) within a transformed space that promotes feature separation. The normal **w** of the separating hyperplane is described as the linear combination of the set of training vectors ($\mathbf{x}_i$, y_i) , $i = 1, ..., N$:

$$\mathbf{w} \ = \ \sum_{i}^{N} \alpha_i y_i f(\mathbf{x}_i) \tag{1}$$

where $f(\cdot)$ denotes the space transformation, the α_i's are the mixing coefficients determined through training, and the y_i's are the labels indicating the class membership of each of the training vectors, $y \in \{-1, \ 1\}$. In practice, most of the α_i's are zero, so the plane is only defined by the so-called *support vectors* (**SV**), which are, roughly speaking, the subset of training samples that are the most difficult to separate (and for which the mixing coefficients are non-zero). For SVM training details, the reader is referred to [4]. Classifying with an SVM amounts to determining on which side of the separating hyperplane a given feature vector lies. This is readily accomplished by projecting the feature vector **x** onto the normal **w** and observing the sign of the resulting projection. Thus, a decision function d_w can be defined as:

$$d_w(\mathbf{x}) \ = \ sgn(\mathbf{w}^T \ \mathbf{x}) = \ sgn\left(\sum_{i}^{N} \alpha_i y_i f(\mathbf{x}_i)^T f(\mathbf{x})\right) \tag{2}$$

so that the classification decision becomes

$$d_w(\mathbf{x}) >= 0 \Rightarrow \mathbf{x} \in C_1 \tag{3}$$
$$d_w(\mathbf{x}) < 0 \Rightarrow \mathbf{x} \in C_{-1} \tag{4}$$

where C_1 and C_{-1} denote the classes associated with labels y_i. In particular, a linear SVM is realized when the spatial transformation is the identity: $f(\mathbf{x}) = \mathbf{x}$. Under such assumption 2 becomes:

$$d_w(\mathbf{x}) = sgn\left(\sum_{i}^{N} \alpha_i y_i \mathbf{x}_i^T \mathbf{x}\right) \tag{5}$$

which is the reference form of the decision function used for this work.

2. Methods

2.1. Platform

The SVM design was implemented on a RC200 development system (Celoxica Ltd., UK), whose core device is a Virtex-II XC2V1000-4 FPGA (Xilinx Inc., USA), with the peripheral support such as:

- Two ZBT SRAM banks
- RS-232 hardware interface
- Four MB video support
- Parallel configuration port

Operating frequency of the system was 50 MHz. The hardware description was written in Handel-C.

2.2. Architecture

The purpose of the hardware SVM design is to evaluate the decision function in (5). The computations involved in this evaluation can be split into the following basic operations:

1. Dot product evaluation; $P = \mathbf{x}_i^T \mathbf{x}$
2. Scalar product evaluation; $S = \alpha_i y_i$
3. Scalar product evaluation; $D = P \times S$
4. Accumulator; $A = A + D$
5. Decision making;

The execution flow controller is then designed to implement the SVM algorithm (Figure 1). At the same time, these basic operations define the main datapath of the design (Figure 2).

2.2.1 Datapath components

To determine the class assignment for a given input vector **x** it is necessary to repeat the first four basic operations as many times as there are support vectors for the implemented classifier. Since logical synthesis of the many required multipliers would impose a high LUT demand on the FPGA [1], the multiplications are mapped to the dedicated fixed-point multipliers (FPM) available in the chosen programmable device. Eight of these elements are used to compute 36-bit multiplications throughout the design; six FPMs perform the dot product (P), while the remaining two perform the scalar products (S,D). The six multiplications required for P are carried out in parallel, while the additions are split into a three-stage 40-bit adder. Since the operations required for S are unrelated to those for P, these are also run in parallel with the computation of P. To avoid losing precision, accumulation A is performed in 48 bits, which seems adequate even for SV counts of 512

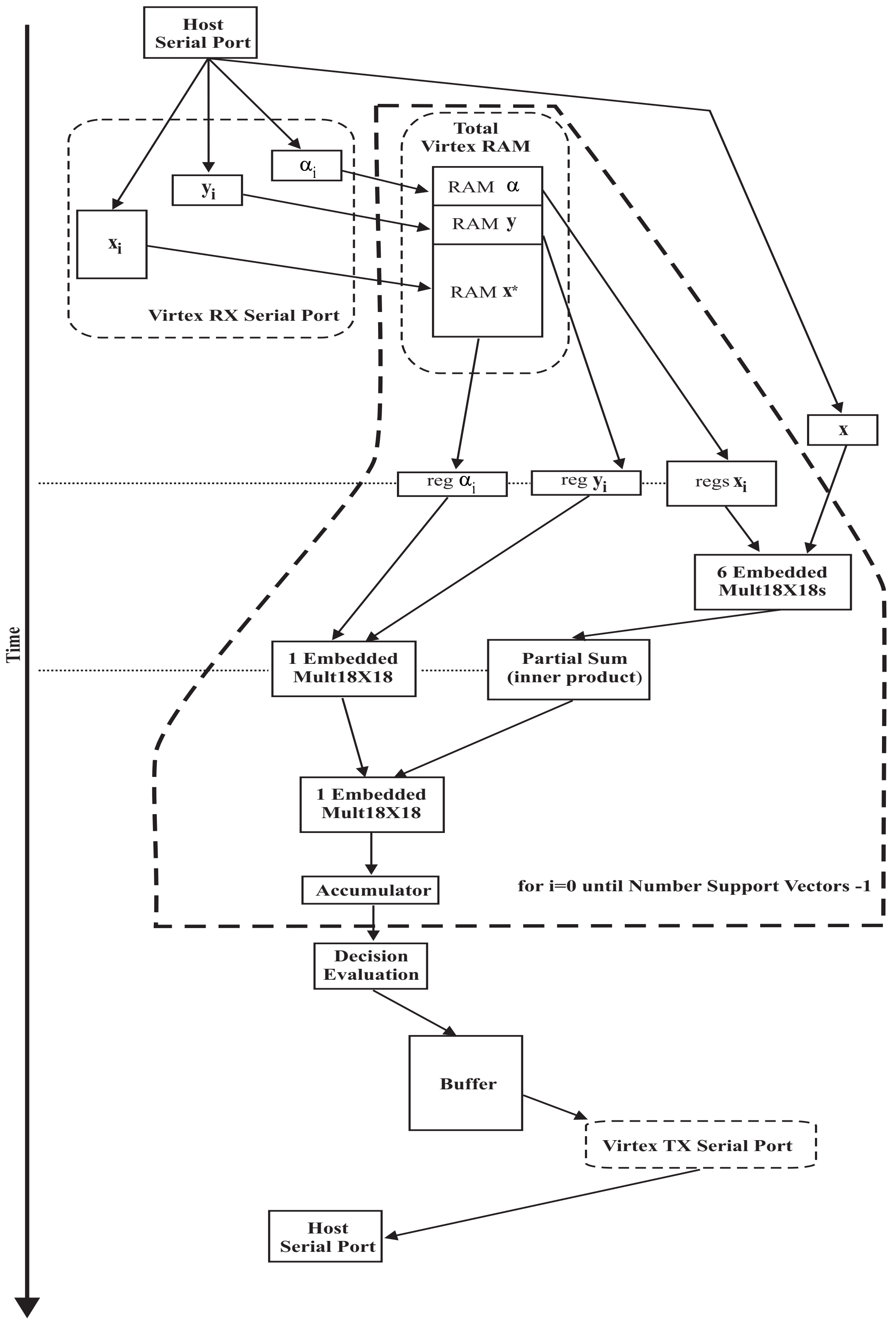

Figure 2. Datapath for the hardware-based SVM system. Vertical line shows the time evolution. Horizontal lines point-out to parallel process. The arrows show the dependences among processes.

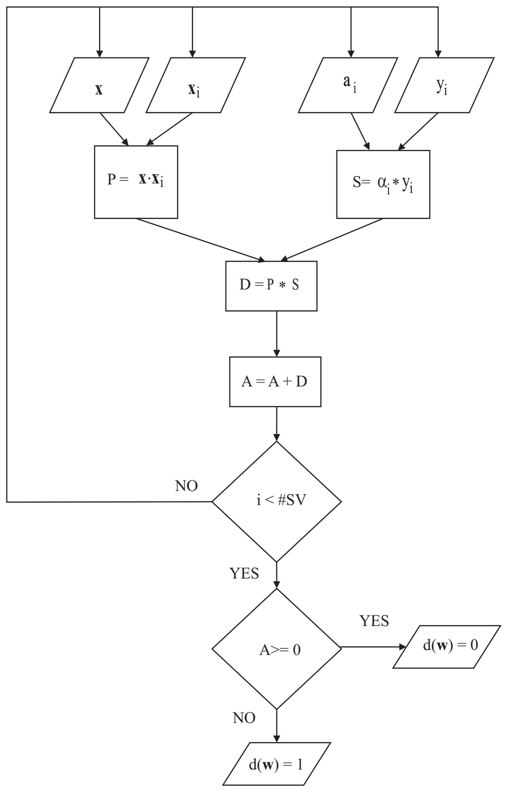

Figure 1. Dataflow diagram of the SVM classification algorithm. See equation (5) and Section 2.2 for variable definitions.

(the maximum allowed in the design). Decision-making is straightforward, since two's complement arithemtic is utilized. The SVM core evaluates a single feature vector at a time, but with the available resources, additional cores could be fit to parallelize multiple classifications.

1. The host serial port sends the support vectors α_i, y_i and $\mathbf{x}_i$
2. The client serial port (Virtex) receives and stores the data into the internal RAM.
3. The hardware waits for a new pattern **x** for classifying.
4. When x is received, α_i and y_i are read from the internal RAM in parallel way. At same time, every component of $\mathbf{x}_i$ is also read.
5. When reading is finished, the product S and the product component by component of **x** and $\mathbf{x}_i$i are done in a parallel way.
6. Finally, the partial products are added in order to obtain the dot product P.
7. The product D=S$\times$P is computed.
8. D is accumulated in A
9. The steps 4 to 8 are repeated for each support vector $\mathbf{x}_i$
10. When the loop is finished, the sign of A is evaluated in order to obtain the class of the pattern x.
11. The classification result is stored into a buffer.
12. The Virtex waits for another pattern to classify, when it is received the steps 4 to 11 are repeated
13. When the buffer is full, the Virtex sends by the serial port the classified datum
14. . The host receives them and store them into a file.

For this implementation the patterns to classify are sent by the serial port from a host system and the classification outcome is also sent to the host in order to downloaded to a file. In a BCI application, the Virtex will read the data from the same embedded system.

2.2.2 Data storage

Parameter vectors for the $\alpha_i,\ y_i,\ \mathbf{x}_i$ are stored in separate internal memory blocks with parallel access. 16-bit precision (Q15, two's complement) was defined for the memory width. While not strictly necessary, feature vectors are also stored in internal memory of the development board.

2.2.3 Host communications

In the current implementation, the feature vectors are originated on a host computer that acquires the relevant experimental data and performs feature extraction tasks. The feature vectors are originated on a host computer that acquires the relevant experimental data and performs feature extraction tasks. Feature vectors are then fed to the SVM core through the serial RS232 interface. Since this interface is eight bits wide, big endian ordering is used for data transmission and storage. Internal access to the $\alpha_i,\ y_i$ banks is full width, i.e., 16 bits. A Java-based comm application was developed for host file access, data formatting, serial transfer control, and file storage of classification results. The serial protocol is carried out at 57600 baud, with one stop bit.

2.3. Validation

In order to evaluate the performance of the SVM architecture, two linear SVMs were designed:

- *SVM 1* was trained to separate synthetic, multidimensional ($\mathbf{x} \in \Re^6$) Gaussian classes with different means and equal variances, using a 1000-sample training set.

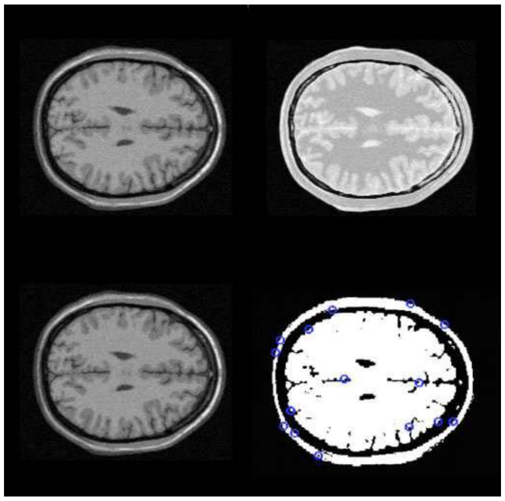
Figure 3. Multispectral MR. top: T1, T2 images; bottom: PD, hardware SVM-segmented images

- *SVM 2* was trained for classification of multispectral magnetic resonance imaging data, obtained from three different excitation sequences: T1, T2, and Proton Density (Fig.3), into white matter and non-white matter voxels. All data came from the same axial slice, corresponding to a set of three images of 256x256 pixels each. Since the SVM architecture was built for six-dimensional features, and only three gray levels are available per voxel, the remaining feature entries were filled with zeros.

In both cases, quadratic optimization training was carried out using the libSVM Matlab extensions [7]. Fixed-point quantization of classifier parameters was performed with the Fixed Point toolbox, also from Matlab. Performance evaluation was carried out using unseen equiprobable sets of 100 test samples for each machine; additionally, the entire multispectral slice was segmented. In both SVM cases, the floating point execution of the trained machines over the test cases (within Matlab) produced 100% correct classification. In order to evaluate the gains in processing time, a high-level simulation of SVM 2, implemented in C using gcc under Linux Mandrake 10 was also executed.

3. Results

The training process of SVM 1 produced a small machine, with only 6 support vectors. Classification performance for SVM 1 is summarized in the confusion matrix of Table 1, which corresponds to 97% correct classification on the test set.

In the case of SVM 2, training produced a support vector set of 78 components, thus requiring more processing time per feature vector. Its corresponding confusion matrix is shown in Table 2, resulting in a classification accuracy of 95%. Full slice segmentation is shown in Figure 3, bottom-right, with white pixels corresponding to white matter structures. In this image, circles indicate voxel locations that were misclassified; the Tanimoto index [8] for this segmentation was 0.9996 (unit Tanimoto index means perfect segmentation), corresponding to 26 misclassified voxels out of 65536.

In regard to processing time, the reference implementation of SVM 2 in C was run on a 550 MHz AMD Athlon machine, delivering a segmented image in 59.5 seconds, 1.84 times faster than the FPGA design, a difference that vanishes when considering that the PC had an 11-fold gain in main clock frequency with respect to the FPGA implementation. Table 3 shows additional time comparisons between the software and hardware versions of this machine.

Finally, Table 4 displays a summary of the FPGA resources utilized by the current SVM 2 implementation.

Table 1. Confusion matrices for SVM 1, both for the floating-point Matlab model and for the fixed-point Virtex-II implemenation.

—	MatLab		Virtex-II	
	C_1	C_{-1}	C_1	C_{-1}
C_1	50	0	49	2
C_{-1}	0	50	1	48

Table 2. Confusion matrices for SVM 2, both for the floating-point Matlab model and for the fixed-point Virtex-II implemenation.

—	MatLab		Virtex-II	
	C_1	C_{-1}	C_1	C_{-1}
C_1	50	0	49	4
C_{-1}	0	50	1	46

4. Conclusion

An FPGA-based implementation of a linear kernel support vector machine classifier has been presented. Given that it is specified that no training has to be carried out in hardware, the design is simple and efficient. Even for the fixed-point constraint, the hardware model achieves a classification rate higher than 95% correct for all the validations performed. Processing time per six-dimensional feature vector was 1.67 ms, for a machine with 78 support vectors, a condition that clearly meets the original design goals for the hardware-based SVM. This time is expected to increase linearly with the number of SVs, and thus an increasingly parallel architecture should be designed (SIMD). The use of

Table 3. Comparison of processing times between the PC simulation and FPGA implementation of SVM 2. t_{TxRx} is total serial transmision time, t_{total} is the entire process duration, t_{FPGA} is net processing time in the FPGA, $t_{vector\ \mathbf{x}}$ is the time for classification of a single feature vector $\mathbf{x_i}$, and t_{VS} is the processing time per support vector

time	FPGA (50 MHz)	PC (550 MHz)
t_{TxRx}	63.7 s	–
t_{total}	173.4 s	59.5s
t_{FPGA}	109.7 s	–
$t_{vector\ \mathbf{x}}$	1.67 ms	0.91 ms
t_{VS}	21.46 μs	11.64 μs

Table 4. Post-synthesis summary of allocated resources for SVM 2

Resource	Available	% Used
Slice Flip Flops	908	8%
Occupied Slices	751	14%
Shift registers	3	–
Bonded IOBs	9	2%
IOB Flip Flops	2	–
Block RAMs	12	30%
MULT18X18s	6	15%
GCLKs	1	6%

Handel-C semaphores, channels and signals with this purpose is currently under investigation.

Every product in the SVM evaluation algorithm was carried out with dedicated multipliers within the Virtex-II chip, as these are single cycle circuits. In the specific case of the scalar product $\alpha_i * y_i$, this multiplication is actually unnecessary, since it basically amounts to a two's complement operation. However, implementing the complement circuit would have unnecessarily increased the overall clock period. Moreover, there might be training solutions that do not provide unit-valued α_i's.

Table 3 demonstrates that the main purpose of this work was succesfully attained. Processing times with respect to the reference software model are in a 2:1 relation, even with the significant variation of main clock frequencies. This indicates that the inclusion of parallel structures in the design would allow its application to heavily demanding tasks. It is not necessary for BCI application an implementation with better performance.

In future work, the serial communication will be replaced by an acquisition system maybe embedded into the same prototype. Indeed, it would be possible that the RC200 kit also performs the stimulation together with the acquisition, classification and feed-backing.

The hardware SVM model proposed and implemented is capable of evaluating any machine that solves a classification problem in six dimensions, since parallelization is currently implemented in the inner product multiplications. El uso de punto flotante o punto fijo, no implica un compromiso entre tasa de buena clasificacin y precisin. Adems el tiempo de clasificacin esta muy por debajo del limite fijado. Esto indica que las optimizaciones al modelo son en sentido del numero de componentes que se utilizan y no en la disminucin del tiempo de clasificacin. No hardware reconfiguration is required for a change of machine as long as the feature dimension does not change. Even for a change in feature space dimension, a simple architectural modification would suffice.

Acknowledgement

Mr. Omar Piña-Ramírez is currently the recipient of scholarship 203532 from the National Council of Science and Technology, CONACyT, Mexico.

References

[1] HDL synthesis for FPGAs. http://www.xilinx.com.

[2] D. Anguita, A. Boni, and S. Ridella. The digital kernel perceptron. *Electronics Letters*, 38(10):445–456, 2002.

[3] D. Anguita, A. Boni, and S. Ridella. A digital architecture for support vector machines: Theory, algorithm, and FPGA implementation. *IEEE Transactions on Neural Networks*, 14(5):993–1009, September 2003.

[4] N. Cristianini and J. Shave-Taylor. *An Introduction to Support Vector Machines and other Kernel-Based Learning Methods*. Cambridge University Press, USA, 2000.

[5] F. Khan, M. Arnold, and W. Pottenger. Hardware-based support vector machine classification in logarithmic number systems. *IEEE Euromicro Symp. Digital System Design (DSD)*, pages 254–261, September 2004.

[6] F. Khan, M. Arnold, and W. Pottenger. Finite precision analysis of support vector machine classification in logarithmic number systems. *IEEE International Symposium on Circuits and Systems*, pages 1–4, May 2005.

[7] C. Lin. libSVM: A library for support vector machines. http://www.csic.ntu.tw/ cjlin/libsvm/index.html.

[8] S. Theodoridis and K. Koutroumbas. *Pattern Recognition*. USA: Academic Press, 1999.