Power Aware Hardware Prototyping of Multiclass SVM Classifier Through Reconfiguration

Rajesh A Patil
MNIT Jaipur
Email:rapatil_rtg@yahoo.co.in

Gauri Gupta
MNIT Jaipur
Email:gauri777@gmail.com

Vineet Sahula
MNIT Jaipur
Email:sahula@ieee.org

A. S. Mandal
CEERI Pilani
Email:atanu@ceeri.ernet.in

***Abstract*—This paper presents power aware hardware implementation of multiclass Support Vector Machine on FPGA using systolic array architecture. It uses Partial reconfiguration schemes of XILINX for power optimal implementation of the design. Systolic array architecture provides efficient memory management, reduced complexity, and efficient data transfer mechanisms. Multiclass support vector machine is used as classifier for facial expression recognition system, which identifies one of six basic facial expressions such as smile, surprise, sad, anger, disgust, and fear. The extracted parameters from training phase of the SVM are used to implement testing phase of the SVM on the hardware. In the architecture, vector multiplication operation and classification of pairwise classifiers is designed. A data set of Cohn Kanade database in six different classes is used for training and testing of proposed SVM. This architecture is then partially reconfigured using difference based approach with the help of XILINX EDA tools. For feature classification power reduction is achieved using reconfiguration.**

***Index Terms*—SVM, dynamic partial reconfiguration, systolic array.**

I. Introduction

Support Vector Machine (SVM) is a powerful machine-learning tool, introduced by Cortes and Vapnik [1]. SVMs are based on the concept of decision planes that define decision boundaries. A decision plane is one that separates between a set of objects having different class memberships. Due to their high classification accuracy rates, in many cases it outperforms well established classification algorithms such as neural networks. Software implementations of SVMs yield high accuracy rates but, they cannot efficiently meet real-time requirements of embedded applications. Hence, SVM hardware architectures emerged as a potential solution to achieve real-time performance. The majority of these emerging architectures is directed towards specific applications and cannot be easily modified to adapt in other scenarios. A significant advantage of SVMs is that whilst neural networks can suffer from multiple local minima, the solution to an SVM is global and unique. Two more advantages of SVMs are that that have a simple geometric interpretation and give a sparse solution. Unlike neural network, the computational complexity of SVMs does not depend on the dimensionality of the input space. Neural network use empirical risk minimization, whilst SVMs use structural risk minimization.

Hardware implementation of any design has its own significant advantages, especially for Image Processing applications, where size of image matrices is quite large. For software implementation, the total simulation and synthesis time are very large, hence hardware implementation is required. Literature suggests that existing general-purpose SVM architectures do not scale well in terms of required hardware resources, complexity, data transfer (wiring) and memory management, primarily because of two important constraints; the number of support vectors (SVs) and their dimensionality. An SVM with a small number of SVs, requires only a few computational modules, however, several applications may require a large number of high dimensional SVs. As such, a general-purpose SVM architecture with a large number of modules faces design challenges such as fan-out and routing, increased complexity, and other performance limitations. An efficient SVM architecture needs to be scalable, provide real-time performance, and be easy to design and implement.

Normally, SVM classifies data into two classes. But we have extended it for six classes using One versus All approach of SVM in which six classifiers are constructed during training. While testing for unknown sample, the class for which the classifier gives highest value of decision function, will be declared as the class of unknown. We classify facial expression into one of the six basic facial expression such as smile, surprise, sad, anger, disgust, fear using six class SVM. Simple hardware architecture for implementation of multiclass SVM classifiers on FPGA is presented. In MATLAB, SVM is trained for six classes, and the extracted parameters are used to implement SVM on the hardware. In the architecture, kernel operation and classification of pairwise classifiers is designed. Cohn Kanade database which consists of image sequences of different facial expressions is used for training and testing of SVM. Difference based approach of partial reconfiguration is applied to this design with the help of EDA tools by XILINX.

The technique presented in this paper depict that after reconfiguration, power (static as well as dynamic) is reduced by a significant amount. Though the work has been done targeting Xilinx Virtex-6 FPGA, the method is general enough to be applied on any FPGA featuring Partial Run Time Reconfiguration (PRTR).

This paper briefly explains feature classification using SVM in section II. In the same section we explain basic principles of SVM and how we are using SVM as classifier for facial expression recognition system. Section III provides a brief overview of other hardware implementations of SVM. Our architecture is presented in section IV. Results are presented

1063-9667/12 $26.00 © 2012 IEEE
DOI 10.1109/VLSID.2012.47

in section V. Finally section VI provide conclusions and future work.

II. Feature classification using SVM

A. Support Vector Machine

Basically SVM is a binary classifier that classifies data into two classes. During the training phase it constructs separating hyperplanes between training samples of two classes, labeled as +1 and -1. The separating hyperplane that best separates the two classes is called the maximum-margin hyperplane and forms the decision boundary for classification. The data samples that lie at the boundary of each class are called support vectors (SVs). During the testing phase only support vectors are involved in the computation. When two data classes are not linearly separable, a kernel function is used to project data to a higher dimensional space (feature space), where linear classification is possible. This is known as the kernel trick and allows an SVM to solve nonlinear problems.

An n-dimensional pattern x has n coordinates, $\mathrm{x} = (x_1, x_2, ..., x_n)$, where each $x_i \in R$ for $i = 1, 2, ..., n$. Each pattern x_j belongs to a class $y_i \in \{-1, +1\}$. Consider a training set T of m patterns together with their classes, $T = \{(\mathrm{x}_1, y_1), (\mathrm{x}_2, y_2), ..., (\mathrm{x}_m, y_m)\}$. Consider a dot product space S in which the patterns x are embedded, $\mathrm{x}_1, \mathrm{x}_2, ...\mathrm{x}_m \in S$. Any hyperplane in the space S can be written as

$$\{\mathrm{x} \in S \mid \mathrm{w}\cdot\mathrm{x} + b = 0\},\ \mathrm{w} \in S,\ b \in R \tag{1}$$

The dot product w·x is defined by

$$\mathrm{w}\cdot\mathrm{x} = \sum_{i=1}^{n} w_i x_i \tag{2}$$

A hyperplane $\mathrm{w}\cdot\mathrm{x} + b = 0$ can be denoted as a pair (w, b). A training set of patterns is linearly separable if at least one linear classifier exists defined by the pair (w, b), which correctly classifies all training patterns. All patterns from class +1 are located in the space region defined by $\mathrm{w}\cdot\mathrm{x} + b > 0$, and all patterns from class -1 are located in the space region defined by $\mathrm{w}\cdot\mathrm{x} + b < 0$. Using the linear classifier defined by the pair (w, b), the class of a pattern x_k is determined with

$$\mathrm{class}(\mathrm{x}_k) = \left\{ \begin{array}{l} +1 \text{ if } \mathrm{w}\cdot\mathrm{x} + b > 0 \\ -1 \text{ if } \mathrm{w}\cdot\mathrm{x} + b < 0 \end{array} \right\} \tag{3}$$

The distance from a point x to the hyperplane defined by (w, b) is

$$d(\mathrm{x}\ ;\ \mathrm{w}, b) = \frac{\mid \mathrm{w}\cdot\mathrm{x} + b \mid}{\parallel \mathrm{w} \parallel} \tag{4}$$

where $\parallel \mathrm{w} \parallel$ is the norm of vector w. Of all the points on the hyperplane, one has the minimum distance $d_{\min}$ to the origin

$$d_{min} = \frac{\mid b \mid}{\parallel w \parallel} \tag{5}$$

We consider the patterns from the class -1 that satisfy the equality $\mathrm{w}\cdot\mathrm{x} + b = -1$ and that determine the hyperplane H_1, the distance between the origin and the hyperplane H_1 is equal to $\mid -1 - b \mid / \parallel \mathrm{w} \parallel$. Similarly, the patterns from the class +1 satisfy the equality $\mathrm{w}\cdot\mathrm{x} + b = -1$ and that determine the hyperplane H_2. the distance between the origin and the hyperplane H_2 is $\mid +1 - b \mid / \parallel \mathrm{w} \parallel$. Hyperplanes H, H_1, and H_2 are parallel and no training patterns are located between hyperplanes H_1 and H_2. So the margin of linear classifier H is $2/\parallel \mathrm{w} \parallel$. The optimum separation hyperplane conditions can be formulated into the following expression that represents a linear SVM, minimize $f(x) = \frac{\parallel \mathrm{w} \parallel^2}{2}$ with the constraints $g_i(\mathrm{x}) = y_i(\mathrm{w}\cdot\mathrm{x}_i + b) - 1 \geq 0, i = 1, ..., m$. The optimization problem represents the minimization of a quadratic function under linear constraints (quadratic programming). A convenient way to solve constrained minimization problems is by using a Lagrangian function. When the Lagrange function is introduced, a Lagrange multiplier λ_i is assigned to each training pattern via the constraints $g_i(\mathrm{x})$. The training patterns from the SVM solution that have $\lambda_i > 0$ represent the support vectors. The training patterns that have $\lambda_i = 0$ are not important in obtaining the SVM model, and they can be removed from training without any effect on the SVM solution. After training, the classifier is ready to predict the class membership for new patterns, different from those used in training. The class of a pattern x_k is determined with

$$\mathrm{class}\ (\mathrm{x}_k) = \left\{ \begin{array}{l} +1\ if\ \mathrm{w}\cdot\mathrm{x}_k + b > 0 \\ -1\ if\ \mathrm{w}\cdot\mathrm{x}_k + b < 0 \end{array} \right\} \tag{6}$$

Therefore, the classification of new patterns depends only on the sign of the expression $\mathrm{w}\cdot\mathrm{x} + b$. To predict new patterns without computing the vector w explicitly we will use the support vectors from the training set and the corresponding values of the Lagrange multipliers λ_i.

$$\mathrm{class}\ (\mathrm{x}_k) = \mathrm{sign}\left(\sum_{i=1}^{m} \lambda_i y_i \mathrm{x}_i \cdot \mathrm{x}_k + b\right) \tag{7}$$

Patterns that are not support vectors do not influence the classification of new patterns. To classify a new pattern x_k, it is only necessary to compute the dot product between x_k and every support vector.

Some common kernels include [2]:

1) Linear : $K(\vec{x}, \vec{z}) = (\vec{x}.\vec{z})$
2) Polynomial : $K(\vec{x}, \vec{z}) = (1 + (\vec{x}.\vec{z}))^d$
3) Sigmoid : $K(\vec{x}, \vec{z}) = \tanh((\vec{x}, \vec{z}) + \theta)$
4) Radial Basis Function :$K(\vec{x}, \vec{z}) = \exp(||(\vec{x} - \vec{z})||^2/(2\sigma^2))$

Binary SVM can be modified for Multiclass classification using following approaches:

1) One Vs One : This method constructs $K(K-1)/2$ classifiers. Classifier ij, named f_{ij}, is trained using all the patterns from class i as positive instances, all the patterns from class j as negative instances, and disregarding the rest. When classifying a new instance each one of the base classifiers casts a vote for one of the two classes used in its training. The class which get maximum votes, will be declared as a class of new instance.
2) One Vs All : OVA method constructs K binary classifiers. Classifier i is trained using all the patterns of class i as positive instances and the patterns of the other classes as negative instances. Test sample is classified in the class whose corresponding classifier has the highest output.

B. Facial expression recognition System

In this system we are using Viola Jones algorithm for face detection and Candide wire frame model, and active appearance model for tracking facial expressions in image sequences and SVM for classification. The first frame of the image sequence is assumed to be of neutral facial expression and last frame corresponds to full facial expression. After face detection, Candide wire frame model is fitted on face image of first frame. Candide model has 113 vertices and 184 triangles. In the subsequent frames as the facial expressions changes, animation parameters of the model will change and model deforms. When we reach to last frame of image sequence, which corresponds to full facial expression, model will be fully deformed. Difference between the model vertices coordinates of first frame and last frame is given as input to SVM [3]. Only geometrical information is given as input to SVM, no texture information is required. Out of 113 vertices only 60 vertices are contributing to facial expressions. So we consider coordinates of only these 60 vertices, in order to reduce the dimensions of the vectors to $60 \times 2 = 120$.

The training phase of SVM produced the following data.

Table I
NUMBER OF SUPPORT VECTORS FOR EACH CLASS

S. No.	Facial Expression	No of Training Samples	No of Support Vectors	Bias Values
1	Smile	25	24	-2.6214
2	Surprise	25	7	-9.3837
3	Sad	21	12	-0.7933
4	Anger	20	26	-3.1861
5	Disgust	23	36	-1.8676
6	Fear	21	87	-0.9999

Class of any test image can be found out by (8), where $K(\mathrm{x}_i, \mathrm{x}_k)$ represent the kernel function. We use polynomial Kernel given by (9)

$$\textit{Decision Function } D(\vec{x}) = \text{sign}\left(\sum_{i=1}^{m} \alpha_i y_i K(\vec{x}, \vec{s}_i) + b\right) \quad (8)$$

$$K(\vec{x}, \vec{s}_i) = (1 + \vec{x}.\vec{s}_i)^d \quad (9)$$

Here $\vec{s}$ represents the support vector (which is independently a column vector) of dimension $120 \times$ (*no of support vectors of any particular class*), and $\vec{x}$ is the test vector which is a row vector of dimension 1×120. α_i is the weight of support vector obtained after training. Here d is considered to be 1.

III. RELATED WORK ON HARDWARE IMPLEMENTATION OF SVM

Hardware implementations of SVMs have gained noticeable interest in recent years, primarily because of the potential real-time performance benefits they offer. 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 [4], an efficient implementation of a kernel based perception is shown in fixed-point digital hardware. The performance of the proposed hardware implementation with the results of a simulated SVM with both fixed and floating point underlying math is compared. Similar performance metrics is obtained 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 [5] 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 [6], 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 is 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 **[7]**.

Most implementations have targeted either specific applications, or require a large amount of computational resources, and thus are not suitable for general-purpose embedded environments. An attempt to design a massively parallel architecture to accelerate learning algorithms was also presented, which utilized arrays of vector processing elements controlled through a host CPU in SIMD fashion. The centralized nature of the CPU, however, creates a bottleneck, emphasizing the need for distributed control. An FPGA implementation utilizing a Microblaze processor as a control unit and custom hardware coprocessors to perform the SVM classification was presented in **[8]**. The design was restricted to 8-dimensional vectors, limiting the application space. An integrated vision system for object detection and localization based on SVM was presented in **[9],** however, the system offered limited scalability. An analog custom processor was presented in **[10]**. FPGA-based coprocessor for Support Vector Machine training and classification in [11], their architecture uses custom low arithmetic precision and is based on clusters of vector processing elements that leverage the FPGA's DSP. In [12] they presented a new systematic approach based on the concept of scalable effort hardware, for the design of efficient hardware implementations for algorithms that demonstrate inherent error resilience. Scalable effort design is based on the identification of mechanisms at each level of design abstraction (circuit, architecture and algorithm) that can be used to vary

the computational effort expended towards generation of the correct (exact) result. SVM implementations have also been explored using embedded processors in **[13]** and **[14]**.

IV. Proposed Approach

Decision function of SVM for classification is given by equation (8). The data flow of this computation is illustrated in Figure 1.

- Kernel Function Calculation: In this calculation two matrices (one is a test vector and another is support vector) are multiplied and a row vector is the output
- In the next step α_i and y_i are multiplied separately.
- Multiplication of α_i and y_i is further multiplied by the kernel function output, producing a scalar value.
- Now this scalar value is added with bias value to produce the final value of the class for which calculation is done.

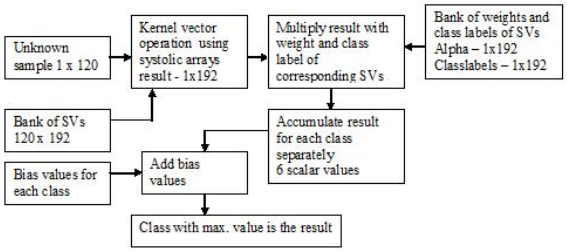

Figure 1. Data flow of proposed approach

The above calculation is done for each and every class. Test Sample $\vec{x}$ belongs to the class for which value of Decision Function is maximum.

Table II
Dimensions for different matrices

S No	Class	Test Vector	Support Vector	Kernel	$\alpha_i y_i$	Decision Function
1	Smile	1×120	120×24	1×24	1×1	1×1
2	Surprise	1×120	120×7	1×7	1×1	1×1
3	Sad	1×120	120×12	1×12	1×1	1×1
4	Anger	1×120	120×26	1×26	1×1	1×1
5	Disgust	1×120	120×36	1×36	1×1	1×1
6	Fear	1×120	120×87	1×87	1×1	1×1

A. Matrix Multiplication using Systolic Arrays

A systolic array is a computing network possessing [15] with the following features: Systolic Arrays are regular arrays of simple Processing Elements (PEs), where each processing element in the array is identical. Systolic algorithm relies on data from different directions arriving at PEs in the array at regular intervals and being combined (This combination may mean any computation like successive multiplication or addition etc.). Systolic Arrays are the architectures preferably used in the matrix multiplication operations. This chain of PEs operates in a pipelined manner and can be expanded vertically or horizontally with minor modifications to operate in a systolic manner. Typical systolic array structure is shown in Figure 2. Block diagram of our proposed systolic array architecture is shown in Figure 3. Total number of support vectors is 192. So we need 192 Processing elements (PEs). In each PE we store one support vector, and its corresponding alpha and class label values. Each support vector has 120 elements. Internal architecture of PEs is shown in Figure 4. In each PE vector operation of SV and input vector is performed, generating a scalar value, which is then multiplied with alpha and class label, simultaneously input vector elements are passed to next PE.

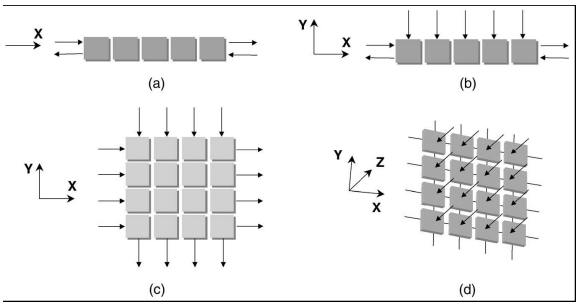

Figure 2. Systolic Array Classification based on Data I/O. (a) Linear array with one-dimensional I/O. (b) Linear array with two-dimensional I/O. (c) Planar array with Perimeter I/O. (d) Planar array with three-dimensional I/O, offering planar data streams with area I/O. [16]

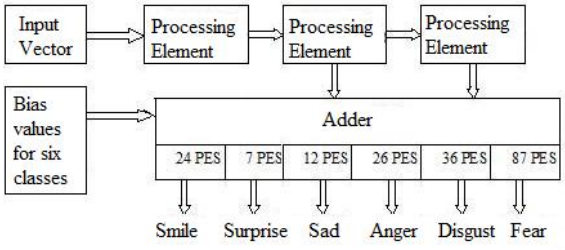

Figure 3. Block diagram of Systolic array implementation

All PEs gives 192 scalar values which are stored in registers. Then class wise these values are added together along with bias values, e.g. first 24 values are added together along with first bias value. Then next 7 values are added together along with second bias values and so on till sixth class. The class which gives maximum value is the class of unknown input vector. Then we perform partial reconfiguration. Partial Reconfiguration is the ability to dynamically modify blocks of logic by downloading partial bit files while the remaining logic continues to operate without interruption. Partial Reconfiguration enables system flexibility, perform more functions while maintaining communication links. Other advantages are size and cost Reduction, time-multiplex the hardware to require a smaller FPGA and power reduction using shutting down power-hungry tasks, when not needed. High Performances, special purpose computer systems are typically used to meet specific application requirements or to off-load computations that are especially taxing to general-purpose computers.

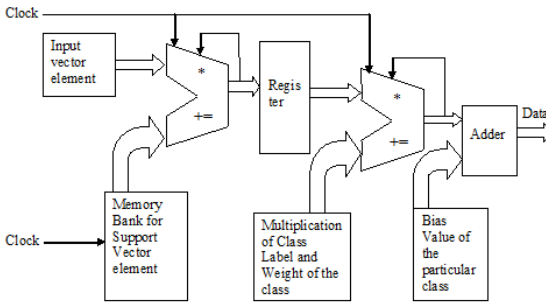

Figure 4. Internal architecture of PE

As hardware cost and size continue to drop and processing requirements become well-understood in areas such as signal processing and image processing, more special purpose systems are being constructed. Because the knowledge gained from individual experiences is neither accumulated nor properly organized, the same errors are repeated. I/O and computation imbalance is a notable example-often, the fact that I/O interfaces cannot keep up with device speed is discovered only after constructed a high speed, special-purpose device.

B. Low Power Reconfiguration Strategy

Partial reconfiguration (PR) is the ability for a portion of an FPGA to be reprogrammed while the remainder of the system remains unchanged. A partial bit stream loads only a portion of the design onto the FPGA rather than rewriting the entire design. Partial reconfiguration is especially useful for reprogramming a portion an FPGA during operation without affecting the rest of the system. This practice is called dynamic partial reconfiguration. Static partial reconfiguration refers to reprogramming a portion of the FPGA while the rest of the board is in a reset state **[17]**. Difference-based partial reconfiguration is used to only make small changes in the FPGA design. The generated bit stream only includes differences between designs. Difference-based partial reconfiguration allows for faster reprogramming of the device since only the changes must be rewritten, but it has limited applications as compared to the module-based solution. Systolic array implementation of SVM classifier is reconfigured as difference based approach. In a system implementing module-based partial reconfiguration, modules that are to be kept in continuous operation without the capability of being partially reprogrammed are referred to as static modules. One or more modules can be designated as the partially reconfigurable module(s), which require additional considerations in the design, synthesis, and implementation stages. Specifically, a modular design flow must be used which will synthesize and create separate bit streams for each of the reconfigurable modules as well as a total system bit stream including all of the static logic and one implementation of each of the reconfigurable modules. Partial reconfiguration can be implemented through a JTAG connection to a PC or internally through custom logic or an on-board processor, such as the embedded PowerPC in the Virtex II Pro FPGA. Partially reprogramming the FPGA through internal circuitry, referred to as self-reconfiguration, is a much more useful method of partial reconfiguration since it eliminates the need for an external PC. The partial bit streams are stored in memory and are written to the Internal Configuration Access Port (ICAP) of the FPGA in order to reconfigure the specified region of the board with the new logic.

C. Reconfiguration Approach and Low Power Implication

Partial Reconfiguration of any design on FPGA leads to save the area and power (both static and dynamic in most of the cases). When design is implemented on FPGA, a particular area is occupied by the logics and different components used in the design. While performing Difference based approach of PR for Systolic Array implementation of SVM Classifier, routing of the nets is changed to reconfigure the particular design on FPGA. This further produces new native circuit description (ncd) file to further produce new bit streams. This partial bitstream is actually the difference of the previous ncd file and new ncd file. When new bit file is loaded to FPGA, using different EDA tools, it is seen very frequently that power (dynamic and static) is lowered than that of before.

V. Implementation and Synthesis Results

All Implementations are done for target device XILINX 6vlx240tff1156-2. Systolic array matrix multiplication is a resource hungry approach because, in each step, a register is required to store the intermediate data. Slices LUTs used is 536 (in number), which can be justified because of large number of registers usage. Also, since register contents are toggling, hence number of LUTs with or without flip flops is significantly large.

Table III
Primitive and Black Box Usage

GND	1	MUXF7	104
INV	4	VCC	1
LUT1	31	XORCY	224
LUT2	196	FD	45
LUT3	11	FDE	192
LUT4	10	FDE_1	224
LUT5	86	BUFGP	1
LUT6	198	OBUF	224
MUXCY	261	DSP48E1	7

Table III shows the primitive and black box usage of LUTs, DSP48s and clocks. Since output value travels to the end PE, hence by table VIth, it can be seen that minimum period is significantly large. Device utilization summary is given in Table IV. Timing summary is given in Table V. Power Optimization is done by using different EDA tools having input as native circuit description files. On chip power summary is given in Table VI. Without reconfiguration, ncd files are obtained using implementation and synthesis tool of XILINX (ISE). Some constraints have been added to produce the switching activity file. These constraints then further be the cause of the power reduction in the design after reconfiguration. Run Time Dynamic Partial Reconfiguration is performed on the design to produce comparison in the total power consumption, before and after the strategy. The total power consumption is the sum

of static power and dynamic power. Power result for Systolic Array Implementation of SVM classifier is obtained by the use of difference based PR terminology. In this terminology, slice logic is changed i.e. routed differently than the original design to partially reconfigure it. Also, some nets are changed to produce new partial bit file, which is the difference of the previous one and the current one. Power supply summary is given in Table VII.

Table IV
DEVICE UTILIZATION SUMMARY

Device: XILINX 6vlx240tff1156-2	Utilized	Available	Percentage Utilization
Number of Slice Registers	461	301440	0.15%
Number of Slice LUTs	536	150720	0.35%
Number of LUT- Flip Flop Pairs with an unused Flip Flop	289	750	38%
Number of LUT- Flip Flop Pairs with an unused LUT	214	750	28%
Number of LUT- Flip Flop Pairs with fully used LUT-FF	247	750	32%
Number of IOBs	225	600	37%
Number of BUFG/BUFCTRLs	1	32	3%
Number of DSP48E1s	7	768	0.91%

Table V
TIMING SUMMARY

Period / Delay	Time (ns)
Minimum Period	6.844 ns
Maximum output required time after clock	0.654 ns

Table VI
ON-CHIP POWER SUMMARY

On-Chip	Power before Reconfiguration (mW)	Power after Reconfiguration (mW)
Clock	3.57	3.57
Logic	2.96	2.96
Signals	2.03	2.03
I/Os	0.18	0.18
DSPs	1.69	1.69
Leakage	1942.17	1921.73

Table VII
POWER COSUMPTION SUMMARY

Power (W)	Total (W)	Dynamic (W)	Quiescent (W)
Before Reconfiguration	2.042	0.010	2.031
After Reconfiguration	2.021	0.010	2.011

VI. CONCLUSIONS AND FUTURE WORK

SVMs exhibit high classification accuracy for different training sets and good generalization performance on data that is highly variable and difficult to separate, making them particularly suitable for expression recognition. Initially hardware description is written for implementation of SVM Classifier. Synthesis result for the design is obtained to have a look in the utilization percentage for different logic circuits and slices etc. on FPGA. Partial Reconfiguration of FPGA device (target board 6vlx240tff1156-2) is then performed. Difference based technique is used to perform partial reconfiguration for Systolic Array implementation of SVM Classifier. Power consumption (static and dynamic) is then evaluated before and after reconfiguration of the design. Power reduction up to 3 to 5 percent has been observed by using XILINX power analyzer EDA tool, due to Reconfiguration of the device. The goal of the technique has been to achieve power reduction due to PR. Hence, metric used for optimization was power. Area and Time also could be a metric for the designs to be performed. In the SVM classification domain, different kernel functions can be used to enhance the accuracy for test samples given for all the facial expressions. Also to perform matrix multiplication operation, faster and efficient methods could be explored in spite of Systolic Array approach. We are modifying the work using module based partial reconfiguration to achieve more power reduction.

REFERENCES

[1] C. Cortes and V. Vapnik, "Support-vector networks," *Machine Learning*, vol. 20, pp. 273–297, 1995.

[2] C. Kyrkou and T. Theocharides, "SCoPE: Towards a systolic array for SVM object detection," *Embedded Systems Letters, IEEE*, vol. 1, no. 2, pp. 46 –49, Aug. 2009.

[3] R. Patil, V. Sahula, and A. Mandal, "Features classification using support vector machine for a facial expression recognition system," in *Springer Machine Vision and Applications. (submitted).*

[4] D. Anguita, A. Boni, and S. Ridella, "Digital kernel perceptron," *Electronics Letters*, vol. 38, no. 10, pp. 445 –446, May 2002.

[5] ——, "A digital architecture for support vector machines: theory, algorithm, and FPGA implementation," *IEEE Transactions on Neural Networks*, vol. 14, no. 5, pp. 993 – 1009, Sept. 2003.

[6] F. Khan, M. Arnold, and W. Pottenger, "Hardware-based support vector machine classification in logarithmic number systems," in *IEEE International Symposium on Circuits and Systems*, vol. 5, May 2005, pp. 5154 – 5157.

[7] ——, "Finite precision analysis of support vector machine classification in logarithmic number systems," in *Euromicro Symposium on Digital System Design*, Aug 2004, pp. 254 – 261.

[8] I. Biasi, A. Boni, and A. Zorat, "A reconfigurable parallel architecture for SVM classification," in *IEEE International Joint Conference on Neural Networks*, vol. 5, July 2005, pp. 2867 – 2872.

[9] R. Reyna, D. Esteve, D. Houzet, and M.-F. Albenge, "Implementation of the SVM neural network generalization function for image processing," in *Proceedings Fifth IEEE International Workshop on Computer Architectures for Machine Perception*, 2000, pp. 147 –151.

[10] R. Genov and G. Cauwenberghs, "Kerneltron: support vector "machine" in silicon," *IEEE Transactions on Neural Networks,*, vol. 14, no. 5, pp. 1426 – 1434, Sept. 2003.

[11] S. Cadambi, I. Durdanovic, V. Jakkula, M. Sankaradass, E. Cosatto, S. Chakradhar, and H. Graf, "A massively parallel FPGA-based coprocessor for support vector machines," in *17th IEEE Symposium on Field Programmable Custom Computing Machines*, april 2009, pp. 115 –122.

[12] V. Chippa, D. Mohapatra, A. Raghunathan, K. Roy, and S. Chakradhar, "Scalable effort hardware design: Exploiting algorithmic resilience for energy efficiency," in *Design Automation Conference (DAC), 2010 47th ACM/IEEE*, june 2010, pp. 555 –560.

[13] R. Pedersen and M. Schoeberl, "An embedded support vector machine," in *International Workshop o Intelligent Solutions in Embedded Systems*, June 2006, pp. 1 –11.

[14] S. Dey, M. Kedia, N. Agarwal, and A. Basu, "Embedded support vector machine : Architectural enhancements and evaluation," in *20th International Conference on VLSI Design*, Jan. 2007, pp. 685 –690.

[15] S. Y. Kung, *VLSI Array Processors*, Mar. 1988.

[16] S. Chai and D. Wills, "Systolic opportunities for multidimensional data streams," *IEEE Transactions on Parallel and Distributed Systems*, vol. 13, no. 4, pp. 388 –398, Apr. 2002.

[17] A. Zeineddini and K. Gaj, "Secure partial reconfiguration of FPGAs," in *IEEE International Conference on Field-Programmable Technology*, Dec. 2005, pp. 155 –162.

[18] S. Rao and T. Kailath, "Regular iterative algorithms and their implementation on processor arrays," *Proceedings of the IEEE*, vol. 76, no. 3, pp. 259 –269, Mar. 1988.

[19] T. Tuan and B. Lai, "Leakage power analysis of a 90nm FPGA," in *Proceedings of the IEEE Custom Integrated Circuits Conference*, Sept. 2003, pp. 57 – 60.

[20] J. Anderson and F. Najm, "Active leakage power optimization for FPGAs," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 3, pp. 423 – 437, March 2006.