Turkish Journal of Electrical Engineering & Computer Sciences

http://journals.tubitak.gov.tr/elektrik/

Research Article

Turk J Elec Eng & Comp Sci
(2016) 24: 3371 – 3387

doi:10.3906/elk-1402-18

Novel dynamic partial reconfiguration implementations of the support vector machine classifier on FPGA

Hanaa HUSSAIN[1,*], **Khaled BENKRID**[2], **Hüseyin ŞEKER**[3]

[1]Department of Electronics Engineering Technology, College of Technological Studies, The Public Authority of Applied Education and Training, Shuwaikh, Kuwait

[2]School of Engineering and Electronics, Edinburgh University, Edinburgh, UK

[3]Bio-Health Informatics Research Group, Department of Computer Science and Digital Technologies, Faculty of Engineering and Environment, The University of Northumbria at Newcastle, Newcastle-upon-tyne, NE1 8ST, UK

Received: 03.02.2014 • **Accepted/Published Online:** 28.09.2014 • **Final Version:** 20.06.2016

Abstract: The support vector machine (SVM) is one of the highly powerful classifiers that have been shown to be capable of dealing with high-dimensional data. However, its complexity increases requirements of computational power. Recent technologies including the postgenome data of high-dimensional nature add further complexity to the construction of SVM classifiers. In order to overcome this problem, hardware implementations of the SVM classifier have been proposed to benefit from parallelism to accelerate the SVM. On the other hand, those implementations offer limited flexibility in terms of changing parameters and require the reconfiguration of the whole device. The latter interrupts the operation of other tasks placed on the hardware device. In this work, two flexible hardware implementations of the SVM classifier are proposed, namely A1 and A2 classifiers with successful applications in a microarray dataset. In addition, two dynamically and partially reconfigurable (DPR) architectures of the SVM classifier are presented. The A1 and A2 architectures have achieved up to 61× and 49× speed-up, respectively, over the equivalent general purpose processor. Furthermore, the DPR implementations achieved at least ∼8× reduction in reconfiguration time compared to non-DPR implementation. This is a significant achievement that can be easily adapted in other application domains of a similar nature.

Key words: Bioinformatics, data mining, field programmable gate arrays, supervised learning, support vector machine

1. Introduction

The support vector machine (SVM) is one of the most recent classes of supervised classifiers and one that has been gaining wide acceptance in recent years [1]. Since its emergence in the early 1990s, the SVM has been used extensively in a vast number of applications such as text categorization, image recognition, and bioinformatics [2]. Bioinformatics applications use SVM in protein homology detection and gene expression. In the latter, SVM is used for molecular classification, such as assigning functions to genes, or in automating the diagnosis and prediction of cancer tissues [3,4]. In many microarray studies, the SVM has shown superior classification performance when compared with other supervised classifiers, mainly due to its amenability to high dimensionality, flexibility in choosing a similarity function, and ability to identify outliers [5,6].

On the other hand, the large dimensionality of microarray data imposes high computational demands on training and classification tasks involved in SVM, which impedes the full exploitation of microarray data in

*Correspondence: hmh.hussain@paaet.edu.kw

formulating complex classification tasks, including in biological studies in the postgenome era. In an effort to counteract the computational limitations of current general purpose processors (GPPs) applied to the analysis of microarray data, two flexible hardware architectures of the SVM classification decision function are proposed in this work using a medium end state-of-the-art FPGA. Additionally, the roles of dynamically and partially reconfigurable (DPR) architectures applied to the proposed SVM are investigated. Accordingly, a collection of five SVM architectures is constructed using combinations of non-DPR and DPR-based SVM cores.

The remainder of this paper will present a background on the SVM classification, followed by relevant work on hardware implementations of the SVM classifier and a proposal of five FPGA implementations of the SVM classifier. The results of these implementations will then be presented and analyzed. Finally, a summary and conclusion will be given.

2. Background

SVMs are supervised machine learning methods that are used to assign a class label to a sample of unknown labels based on a prediction model constructed using a set of data of known class labels; this is called learning or training of data.

The SVM classification consists of two discrete phases; one is the training phase and the other is the evaluation of the decision function, known as the classification phase. During the training phase, the SVM estimates a function that classifies the data into two classes by forming a hyperplane that maximizes the separation of the two classes [5]. The SVM deals mainly with problems of binary classes (class label = 1 or class label = –1), and when multiclass problems are used, the SVM is applied to two classes at a time until all classes are covered. During the training phase, data are mapped on to a large feature space where the classifier tries to estimate a multivariate function from a given training set that can separate the two classes by constructing a hyperplane that maximizes this separation [2,3].

Given a training set $(\mathrm{x}_i, \mathrm{y}_i)$, where i = 0 to N – 1 (N is the number of training samples), $\mathrm{x}_i \subseteq \Re^M$ are the training features, M is the number of features or dimensions, and $\mathrm{y}_i \in \{-1,1\}$, being the known classifications of the i training sample, the classification function of linearly separable training data is given by Eq. (1) [2]:

$$f(x) = \langle w.x \rangle + b, \tag{1}$$

where b is the bias or distance between the hyperplane and the origin, and **w** is a normal vector of the separating hyperplane. The hyperplane seeks to maximize the distance between the two soft margins by minimizing the norm **w** for linearly separable training. Solving an optimization function leads to a kernel notation shown in Eq. (2):

$$L(\alpha) = \sum_{i=0}^{N-1} \alpha_i - \frac{1}{2} \left[\sum_{i,j=0}^{N-1} \alpha_i \alpha_j y_i y_j K(x_i, x_j) \right], \tag{2}$$

where $\mathrm{K}(\mathrm{x}_i, \mathrm{x}_j)$ is the kernel function, which can be linear, Gaussian, or polynomial, as formulated in Eqs. (3a), (3b), and (3c) respectively:

$$k(x_i, x_j) = x_i.x_j, \tag{3a}$$

$$k(x_i, x_j) = e^{\left(\|x_i - x_j\|^2 / 2\sigma^2\right)}, \tag{3b}$$

$$k(x_i, x_j) = (1 + x_i.x_j)^p. \tag{3c}$$

The linear SVM classifier is used in the hardware implementations proposed in this work; consequently, the linear kernel function is used, leading to transforming Eq. (1) to Eq. (4).

$$f(x) = \sum_{i=0}^{N-1} y_i \alpha_i (x_i . x_j) + b. \tag{4}$$

For simplifying the hardware implementations, bias b can be set at zero assuming that the hyperplane is passing through the origin, and the query x_j will alternatively be labeled as Q_j during the classification phase to distinguish it from the support vectors x_i obtained during the training phase. Note that the x_i used in the training phase represents the complete training samples, whereas x_i used during the classification phase represents the support vectors (SVs). The latter forms a subset of the training samples having nonzero α_is. During the classification phase, when the SVM classifier is presented with a query vector Q_j, it performs the classification function defined in Eq. (5) based on the linear kernel to determine in what side of the hyperplane the query lies [5].

$$\text{Query Class } (Q) = sgn\left(\sum_{i=0}^{N-1} y_i \alpha_i x_i^T Q\right) \tag{5}$$

In this work, the FPGA implementation tries to determine Eq. (5) given that the training phase is done offline and that the core is supplied readily with the SVs.

3. Relevant work

The SVM has established itself as a superior supervised classification method in a wide range of applications [2–6]. As a result of its popularity, many efforts have been expended towards the acceleration of the SVM and the enhancement of its real-time performance using FPGAs. FPGA implementations of SVM are directed toward three main areas, namely accelerating the training phase, accelerating the classification phase, or accelerating both in a single architecture. The following overview is a selection of relevant FPGA implementations of the SVM classifier.

The earliest work reported in the literature on FPGA implementation of SVM training was in [7], targeting nonlinear classification. The authors proposed and implemented a digital architecture of SVM in FPGA targeting the learning or training phase only. The architecture consisted of two main parts, the first to solve the constrained quadratic problem given an initial constant bias and the second to iteratively update this bias. The authors tested the architecture on Xilinx Virtex-II for the case of 8 and 32 patterns, achieving an acceptable rate of classification for both [7]. The work did not include acceleration results with respect to the GPP and it was mainly focused on proving the suitability of the application to its hardware implementation.

The authors of [7] have various subsequent works in this area. One of their recent works was reported in [8], where they presented an FPGA core generator tool aiming for automatically generating a Gaussian kernel SVM architecture in VHDL. The user enters the desired parameters to the graphical user interface (GUI), the number of support vectors, number of features, data rate, gamma variable required for the computation of the Gaussian kernel, and precision of the input data. The core then customizes the hardware description accordingly to solve the Gaussian classification function [8].

In [9], the authors reported architecture of the SVM classifier that performed the training phase based on sequential minimal optimization (SMO). The main contribution of the work was to implement the SMO-

SVM using DPR, whereby the modular blocks performing the tasks associated with SVM training were time-multiplexed, leading to an area saving of 22.38% of the design implemented in the Xilinx Virtex-4 XC4VLX25 FPGA.

In [10], the authors reported a hardware implementation of the SVM classifier that performs both training and classification on a FPGA based on three types of kernels, linear, Gaussian, and polynomial, using a recursive-updating equation. The architectures targeted disease diagnosis based on using microarray data. The authors tested their architecture on sonar and cancer data, achieving superior classification performance in terms of classification and accuracy, especially with the linear kernel. The authors mentioned that the area footprint was one of the limitations of the design; however, they did not report the total CLB slices consumed by the architecture. In addition, the implementation attained a low clock speed of 25 MHz for the leukemia implementation.

In [11], the authors presented FPGA implementation of the SVM classifier targeting a brain–computer interface, which requires real-time decisions. The design depends on doing the training offline using the LibSVM MATLAB extension based on linear kernels and the training coefficients along with the SVs are made available to the architecture. The architecture realized the classification decision function based on parallelizing the computation of the linear kernel. In addition, it was based on processing six dimensions in parallel using embedded 18×18 multipliers on the Xilinx Virtex$-$II XC2V1000-4 FPGA. The architecture performed well in terms of classification when compared with a floating point implementation. However, for one of the cases, FPGA performed worse in terms of processing speed, consuming twice more time than the GPP. In addition, the FPGA architecture was nonscalable, limiting the implementation to six dimensions only [11].

As a consequence of the widespread use of the SVM classifier in various applications, many published works have been reported in the literature on several high-performance FPGA implementations. The reader is advised to consult [12–18] for details about some of these additional works.

In addition to being a popular algorithm for hardware implementation, the SVM has been incorporated in many data mining software suites such as the LibSVM tool [19], MATLAB, the R statistical package, SVM Fu, SVM Torch, and many others.

The SVM architectures presented in this work build on existing ones and try to overcome their main limitations in terms of scalability and flexibility. The main contribution of the work presented here is the application of DPR to enhance the flexibility toward a more adaptive SVM core. This feature is intended to cater to the variability in microarray studies. In addition to using DPR, our implementation is different from most of the above in being parameterized, which means it can adapt to different data sizes.

4. Novel FPGA implementation of the SVM classifier

A total of five SVM architectures are presented. Two of the proposed architectures of the SVM classifier are based on systolic array architectures, designed to target different dataset sizes; however, both of them work toward determining Eq. (5). The first architecture targets problems where the dimension (M) is greater than the number of SVs, while the other is designed to solve the opposite case. The architectures are labeled A1 and A2, respectively; architecture A1 was reported by the authors in a previous work [20]. Both architectures consist of modular designs captured in Verilog HDL having four main blocks. However, the design of each block and the interconnection between the internal elements in A1 architecture vary from those in A2. The first block is the memory, which is responsible for storing the training data. The second block is responsible for the computation of the linear kernel using the training data, class labels, and coefficients received from the memory

block. The third block is responsible for accumulating the results coming from the kernel computation block as they get computed. Finally, the decision-making block receives the result from the accumulation block when all data in the training set have been processed to determine the final result, which represents the class label of the query. The following subsections will provide more details on each block for the two proposed architectures.

4.1. Memory block

4.1.1. A1 architecture (M >>SVs)

This block stores four types of data describing the training set, thus comprising four memory subblocks as shown in Figure 1a. The first memory subblock stores the complete training set in the form of a matrix of a number of rows equal to the number of SVs and a number of columns equal to the number of features or dimensions (M). These are implemented as a set of FIFOs of a number equal to the number of SVs having a depth of M each, and a width of B (WL of each feature). The second memory subblock is a FIFO used to store the class labels of the given SVs. When these SVs are small, the associated labels are stored inside the FPGA internal registers instead of wasting a complete block RAM due to the fact that each class label requires one bit only

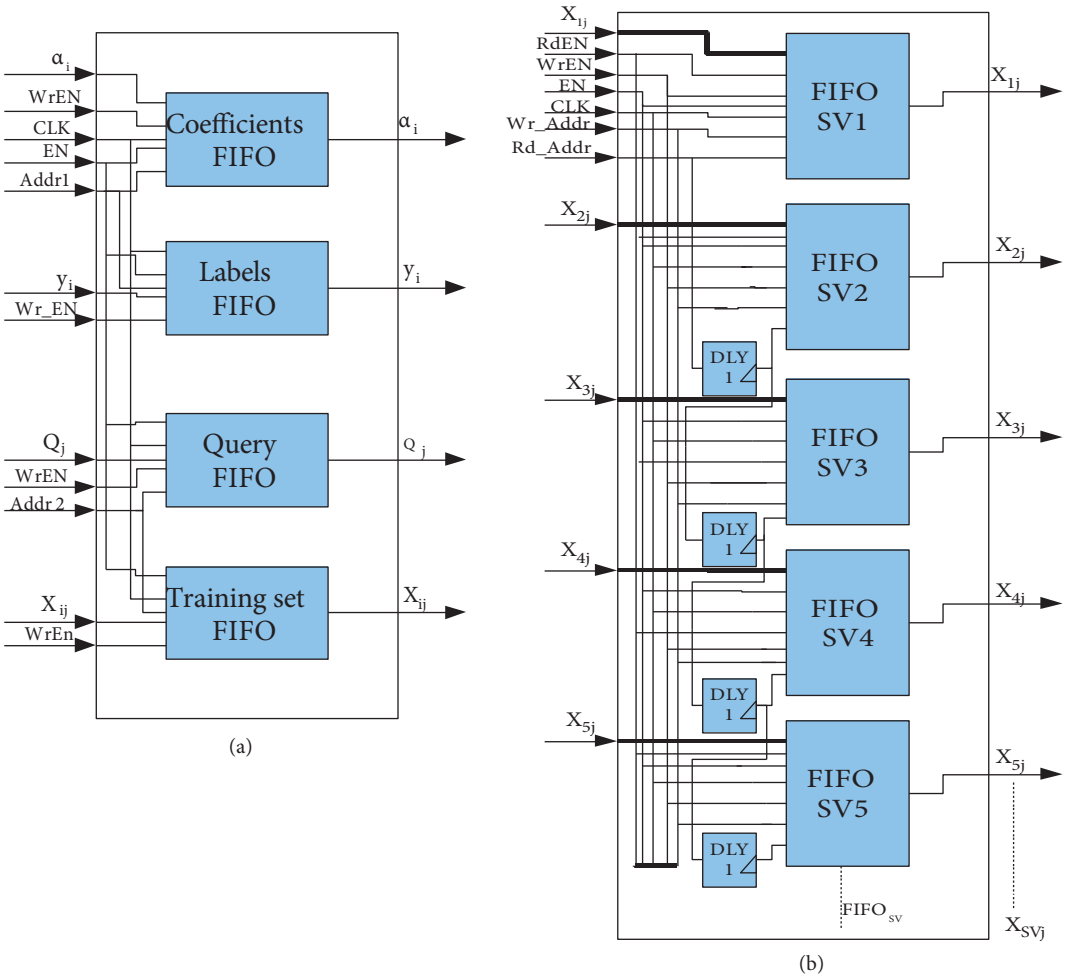

Figure 1. The data path of the memory block of the SVM classifier: (a) the components constituting the memory block, highlighting that there are four different storage subblocks to service the SVM classifier; (b) the components of the memory responsible for storing the training set in SVM architecture A1.

(to represent binary class labels). Since M is expected to be significantly large for this architecture, the third memory subblock is used as a FIFO to store the features of the query. Note that multiple queries can be stored in this memory if enough block RAMs are available. As for the fourth memory subblock, it is used to store the training coefficients (α_is) computed offline during the training phase and has a depth equal in number to the SVs.

The complete block is scalable in terms of the parameters B, M, and SVs. This block is connected to the kernel computation block, the latter consisting of kernel processing elements (Kernel PEs). Each of the SV-FIFOs instantiated by the core design to store the training set is allocated to one of the Kernel PEs, to supply each Kernel PE with one feature every clock cycle in a pipelined manner. Additionally, one feature of the query FIFO is read by the first Kernel PE every clock cycle and propagated through the pipeline, allowing for parallel SV kernel computations. On the other hand, the class labels are read from the memory every clock cycle after a latency of M clock cycles needed to fetch the first kernel result. Similarly to the training coefficients, they are read after M clock cycles, as they are required at the same time that the class labels are needed for the completion of the kernel computation (as will be illustrated in subsequent subsections).

The architecture of the memory storing the training set is the most sophisticated part within the memory block as it involves adaptive techniques to instantiate multiple FIFOs, each responsible for storing the whole dimensions of one SV (1 SV is a vector of M features). As such, the number of FIFOs is equal to the number of SVs. Since these FIFOs are designed to feed the kernel computation systolic array, the read address from these FIFOs is pipelined throughout the SV FIFOs to ensure that data are read by the corresponding kernel PEs in a timely manner as illustrated in Figure 1b.

4.1.2. A2 architecture (SVs >>M)

The components and functions of the memory block are similar to those of the A1 architecture. However, the specifications, arrangement, and number of FIFOs are different since they serve the case when the number of SVs is much higher than M. In A2 architecture, the training coefficients are stored in FIFO, having a depth equal to the number of SVs and width equal to the WL of each coefficient. As for the class labels and query memory, the tool is left to decide whether to store the data inside registers or utilize block RAMs based on the preset M and SVs. Memory handling of the training data consists of M-FIFOs, each having a depth of SV, where each FIFO is associated with a specific kernel PE pipelined similar to A1 architecture to serve the requirement of the kernel computation block.

In summary, the depth of the training set memory in each of the two architectures is different: in A1, the depth is M, while in A2, it is SV. On the other hand, the number of FIFOs to store the training set for A1 and A2 is SV and M, respectively.

4.2. Kernel computation block

4.2.1. A1 architecture

The kernel computation block is partitioned into three subblocks operating in two stages as shown in Figure 2, whereby each subblock is pipelined to perform a portion of the computation. During the first stage, the largest and most time-consuming work is carried out by the subblock referred to as Multiplier A to compute the dot product (linear kernel) in Eq. (6):

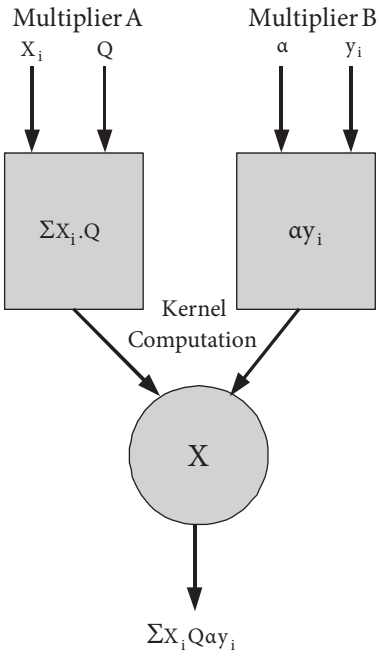

Figure 2. Data path of the kernel computation block.

$$\text{Multiplier } A = \sum_{j=0}^{M-1} x_j Q_j, \tag{6}$$

where x_j is a SV feature and Q_j is the corresponding query feature. This subblock consists of a systolic array of a number of SV kernel PEs, where each PE has the role of receiving one SV feature every clock cycle (x_{ij}) along with the corresponding query feature (Q_j). While each PE has a local FIFO associated with it whose sole responsibility is to provide that particular PE with one SV feature every clock cycle, only one Query FIFO is commonly used by all PEs as explained previously. The architecture of the systolic array of this subblock is shown in Figure 3a. The first PE in the array reads a query feature every clock cycle from the Query FIFO and propagates that feature throughout the pipeline. Figure 3b illustrates the functionality of each PE, where each PE processes all the features of one SV independent from each other, except for the pipelining of the query features.

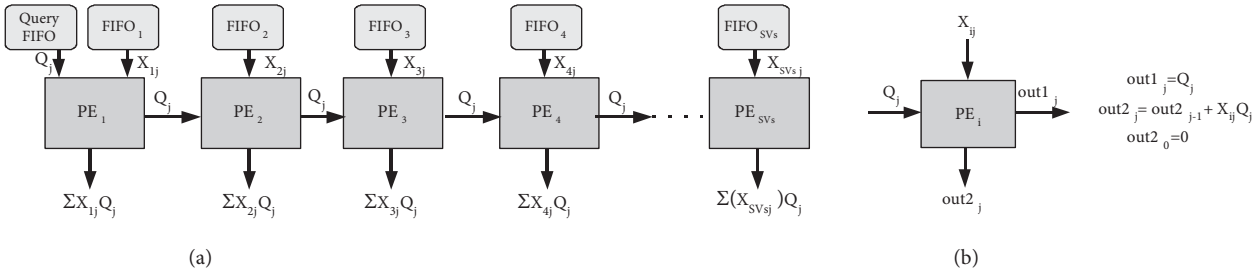

Figure 3. (a) The systolic array of Multiplier A1 of the kernel computation block; (b) the functionality of a single PE.

The systolic array is fully parallelized such that the SV computations of Eq. (6) are carried out simultaneously. This operation is facilitated by the capability to obtain the needed feedstock for each PE

continuously from the local memory attached to each PE. The latency of the pipeline is M clock cycles, while the throughput is one result per clock cycle corresponding to the multiplication shown in Eq. (6) for one SV. Consequently, for processing one query vector, M + SV clock cycles are required by the pipeline to finish the computation.

Additionally, while the above computations are being carried out by the first subblock, the second subblock, referred to as Multiplier B, which is associated with the first stage of the kernel computation block, starts operating just after a period of M – 1 clock cycles. This delay is required to ensure appropriate synchronization with the previous and subsequent classifier operations, and to ensure efficient propagation of the data throughout the pipeline. The function carried out by this subblock is to read the training coefficients and the class labels associated with each SV simultaneously from the memory block and compute the scalar product shown in Eq. (7):

$$\text{Multiplier } B = \alpha_i y_i, \tag{7}$$

where i is from 0 to N – 1. To complete the computation of the kernel for a single SV, the subblock in stage two multiplies the two results of stage one (results of Multipliers A and B) to obtain the final kernel result given by Eq. (8).

$$\text{Kernel Computation } = \sum_{j=0}^{M-1} X_{ij} Q_j \alpha_i y_i \tag{8}$$

It can be stated that the last multiplier is pipelined with the previous two such that it starts working after M clock cycles (the latency of the first subblock), and thereafter it outputs one result per clock cycle. In this work, the multiplication operation carried out in the three subblocks is performed using DSP48 blocks available from the Xilinx Virtex-4 FPGAs and subsequent series.

4.2.2. A2 architecture

The components of the kernel computation block of the A2 architecture of the SVM classifier are similar to the SVM A1 architecture. However, the difference lies in the arrangement and number of PEs constructing the systolic array of Multiplier A. The systolic array now scales with M instead of SVs in A1 architecture, where M PEs constitute the systolic array responsible for computing Eq. (6), each having a local memory attached to it having a depth of SV as was explained in the memory subsection.

This architecture serves the case when SVs >>M, parallelizing the computation of the kernel partial product by allowing M computations to be carried out simultaneously. On the other hand, the query M features are stored in registers within the subblock as they are needed by the PEs every clock cycle. Each of the M PEs has the role of receiving a feature from one SV every clock cycle and propagating the partial product result to the next PE in the pipeline to complete the computation of Eq. (8) for the same SV. Unlike the A1 architecture, the computation of Eq. (8) for one SV is partially carried out by each PE, and the result of the last PE of the systolic array will correspond to the final result of Eq. (8). The latency of this subblock is M clock cycles, and the throughput is one result per clock cycle. Consequently, the time needed for this subblock to complete the computation of the kernel's partial product is the same as in A1 architecture, being M + SVs. Figure 4 outlines the arrangement and functionality of A2 architecture.

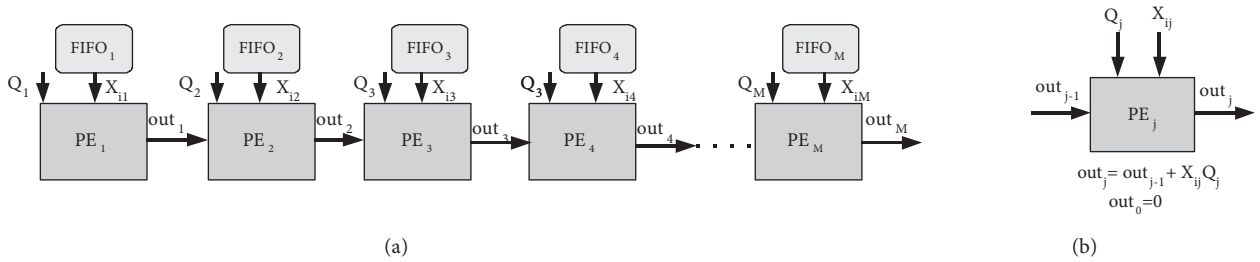

Figure 4. (a) Systolic array architecture of the first subblock in the kernel computation block for SVM A2 architecture; (b) the functionality of each PE.

Lastly, the computation of the remaining parts of the kernel is activated in a similar way to A1 architecture, whereby Multiplier B starts working after a period of M – 1, such that after M clock cycles the results from both subblocks of stage one are ready and synchronized, allowing the subblock of stage two to start receiving results.

4.3. Accumulation block

This block is a simple add-and-accumulate circuitry required to accumulate the results as they come in from the kernel computation block, as shown in Figure 5. The final accumulation result is given by Eq. (9).

$$\text{Accumulation result} = \sum_{i=0}^{SVs-1} \left(\sum_{j=0}^{M-1} X_{ij} Q_j (\alpha_i y_i) \right) \tag{9}$$

4.4. Decision-making block

This block is the simplest circuit in the whole SVM classifier common to both architectures A1 and A2, whose role is to determine the class label of the query based on the sign of the accumulation result given in Eq. (9). The SVM classifiers proposed in this work target binary class labels, e.g., class zero distinguishes a diseased tissue and class one is healthy tissue. This block basically checks the most significant bit (MSB) of the final accumulated results, whereby the query is assigned to class label of 1 when the MSB is 1, and to zero otherwise.

5. Novel DPR implementations of the SVM classifier on FPGA

This section presents two DPR implementations of the SVM classifier based on A1 and A2 architectures. DPR is basically a feature in modern FPGAs that enables users to set portions of the tasks placed on the FPGA as reconfigurable tasks while others are static. This feature allows users to alter the configuration of any of the reconfigurable tasks while the device is running without affecting the operation of the static tasks. This capability enables swapping existing tasks on the FPGA with completely different tasks or swapping the existing tasks with variable copies of the same tasks having different parameters, as with the case in the subsequent implementations. DPR follows a specific design flow and requires a specialized software tool.

5.1. Novel DPR implementations based on single-core SVM classifier

Two reconfigurable single-core implementations of the SVM classifier are presented here. The first reconfigurable single-core SVM classifier is based on an A1 SVM classifier. The single core is used to construct a partial reconfiguration (PR) design using the Xilinx PlanAhead tool. The complete SVM core is set as a reconfigurable

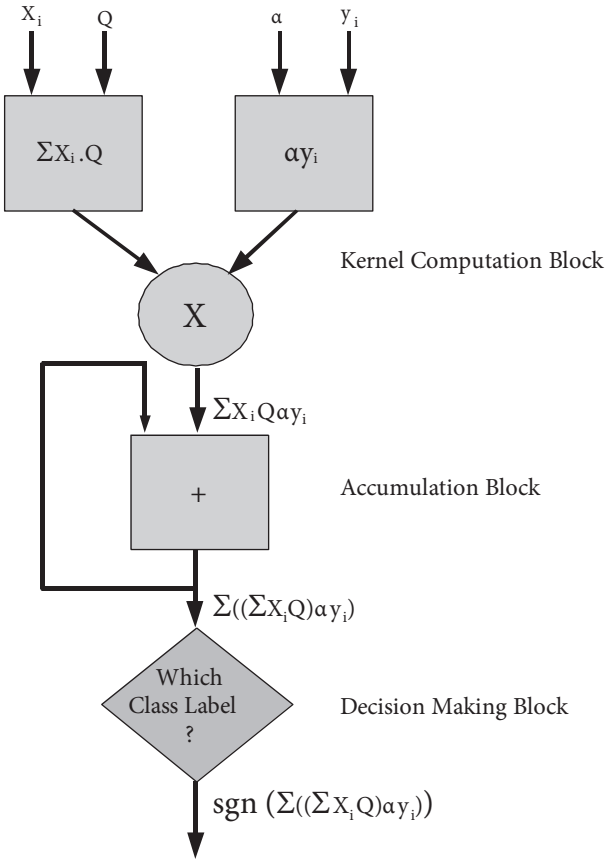

Figure 5. The data path of the complete SVM classifier applicable to A1 and A2 architectures, including the functionality of the accumulator and decision-making block.

partition (RP) following Xilinx PR design flow and hierarchical methodology [21], and is used to create multiple reconfigurable modules (RMs). RMs are basically variant copies of a portion of the classifier or a complete classifier that can be swapped in and out of the FPGA on the fly to alter the operation of that portion. The second reconfigurable single core is based on setting one core within a multicore SVM architecture as a reconfigurable core. As such, a quad-core SVM classifier is constructed using the normal design flow and then is used in this implementation whereby one of the quad cores is made reconfigurable only while the others are left static. The implementation is based on the following parameters for each core: B = 9, SV = 20, and M = 1024. Results of the two implementations will be presented in the subsequent sections to assess the advantages of these implementations in terms of reconfiguration time, placement, and flexibility.

5.2. Novel DPR implementation based on multicore SVM classifier

To enhance the aforementioned multicore architecture and add more flexibility to it, a DPR implementation based on setting all SVM classifiers within the quad-core as RPs is proposed. The flexibility added here is in altering the contents of the memory, relocatability of the cores, and processing of multiple data in parallel using different parameters to conduct multiple studies. The relocatability feature is crucial in server solutions where users may request cores to be activated on demand; consequently, rearranging existing cores within the same chip is necessary. In addition, this feature allows for relocating cores from defective FPGA fabric to healthy locations.

After constructing the quad-core SVM classifier, each of the quad cores is set as a RP at first. Second, multiple variants of each core corresponding to different memory contents, SVs, M, and B are created, which will be used in creating multiple configurations as illustrated in Figure 6.

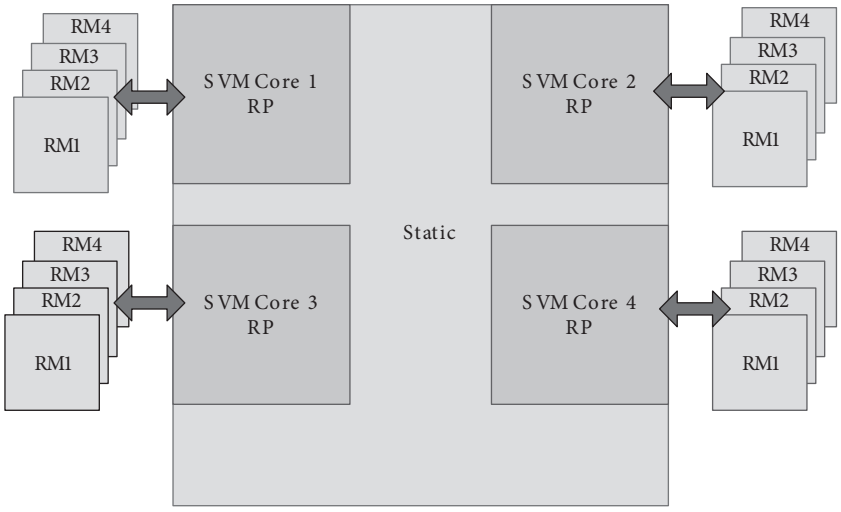

Figure 6. Block diagram illustrating the reconfigurable quad-core SVM classifier.

Third, the quad RPs are constrained within specific regions on the FPGA containing all the logic and resources required to account for the maximum parameters chosen in any of the required RMs. The bit streams of several configurations are then created corresponding to variable copies of the SVM classifier. This process is associated with the generation of two bit streams per configuration; one is a full bit stream that can be used to configure/reconfigure the whole FPGA device, while the other is a partial bit stream that is used to partially reconfigure any of the quad cores during run-time as required.

6. Implementation results

This section presents the results of the aforementioned implementations, which include results from both hardware and software implementations of the SVM classifier. In the hardware, used data are of different sizes that can be stored within the block RAMs of the selected FPGA available onboard the Xilinx ML 403 platform board. The software implementations on the GPP are based on using the MATLAB (R2009b) bioinformatics toolbox running on a 2.60 GHz Pentium Dual-Core E5300, with 3 GB RAM workstation. The toolbox includes an optimized SVM classification function that can be easily utilized in addition to using the fixed-point toolbox to quantize the features according to the selected precision and required integer WL. The following subsections summarize the implementation results.

6.1. Single-core SVM classifier based on A1 architecture

The proposed single-core A1 architecture of the SVM classifier was simulated first using synthetic data that mimic a microarray dataset, then synthesized, mapped, placed, and routed using Xilinx ISE 12.2 to target the XC4VFX12 FPGA available onboard the Xilinx ML 403 platform board. The implemented design was based on these parameters: B = 8, M = 1024, and SVs = 20.

The hardware implementation was tested using the Xilinx ChipScope Pro Analyzer 12.2 and checked against simulation results. The number of clock cycles to classify one query was found to be 1048, achieving an

execution time of 10.62 μs based on the attained frequency of 98.7 MHz. On the other hand, the execution time of the GPP implementation was 646 μs based on taking the average of 10,000 runs. Consequently, the FPGA implementation of the A1 architecture outperformed the GPP implementation by approximately 61 times. The resources used by the FPGA for this implementation are shown in the Table.

Table. The FPGA resources used by A1 and A2 architectures.

Device	Xilinx XC4VFX12-12ff688			
Parameters	A1: B = 8, M = 1024, SVs = 20		A2: B = 8, M = 20, SVs = 1024	
	Used/available	Utilization rate (%)	Used/available	Utilization rate (%)
Slices	1703/5472	31	1206/5472	27
Slice FF	2137/10,944	19	1810/10,944	17
4 input LUTs	1799/10,944	16	1705/10,944	15
Block RAMs	23/36	63	21/36	58
DSP48	22/32	68	21/32	65
Clock frequency	98.7 MHz		142.9 MHz	

The same design was simulated using a higher end FPGA, namely Xilinx XC4VSX35, achieving a frequency of 137.7 MHz, which led to hardware execution time of 7.64 μ s, consequently achieving a speed-up of $\sim$85 times over an equivalent GPP implementation. This finding was based on simulation results only; actual implementation onboard was not feasible due to the unavailability of the large hardware device.

6.2. Single-core SVM classifier based on A2 architecture

Following the same procedure above with the A2 architecture of the SVM classifier, the design was implemented with these parameters: B = 8, SVs = 1024, and M = 20. The number of clock cycles to classify one query was found to be 1048, which was the same as the previous block, as was expected since the two architectures require the same number of clock cycles to classify one query. However, the attained clock frequency of the A2 architecture was 142.9 MHz, which was higher than A1, leading to an execution time of 7.34 μ s. On the other hand, the execution time of the GPP implementation was 359 μ s based on taking the average of 10,000 runs. Consequently, the FPGA implementation of the A2 architecture outperformed the GPP implementation by approximately 49 times. The resources used by the FPGA for this implementation are shown in the Table.

It is difficult to compare two different technologies with each having its own architecture and components. However, the two technologies used in the aforementioned comparisons were bought at the same time and both are considered to be medium end devices, which ensures a fair comparison. Current state-of-the-art Virtex-7 FPGAs have much larger logic resources ranging from 91,050 to 136,900 slices as compared to the 5472 slices available in the device used in this work. However, such FPGAs are expensive and not widely available on the market. As for the GPPs, current state-of-the-art multicore Intel processors have between 2 and 10 cores. Therefore, using a dual core Intel processor seems to be fair choice when compared with the size and scale of the FPGA used in this work.

6.3. Effect of data dimensionality: GPP vs. FPGA

The FPGA implementation of the A1 SVM classifier was compared with GPP implementation running on the MATLAB (R2009b) bioinformatics toolbox to particularly investigate the effect of changing the number of dimensions M on the timing performance of the two implementations. The number of SVs was fixed at 1024, B was fixed at 16, and M was varied from 1 to 16. Figure 7a shows that when M was increased, both

implementations took longer times; however, the FPGA suffered less in classification time as compared with the GPP implementation. For instance, changing M from 1 to 16 features increased the classification time by only 1.4% for the case of FPGA implementation as compared to 5.7% for GPP implementation. Figure 7b implies that increasing the dimensions of the SVs yields a significant increase in the classification time of the GPP implementation as compared to a smaller increase for the FPGA case as shown in Figure 7a. Thus, the GPP implementation seems to scale linearly with increasing dimensions while the FPGA implementation seems to be nearly constant. The possible reason for the GPP behavior is that increasing the dimensionality of the SVs beyond those shown in Figure 7 causes the multiply-accumulate operations involved in the kernel computation phase implementation to access the global memory to store the intermediate results since the cache will not be able to hold these values. On the other hand, the FPGA implementation still maintains its performance due to the abundant on-chip storage resources (block RAMs) and the use of fixed precisions.

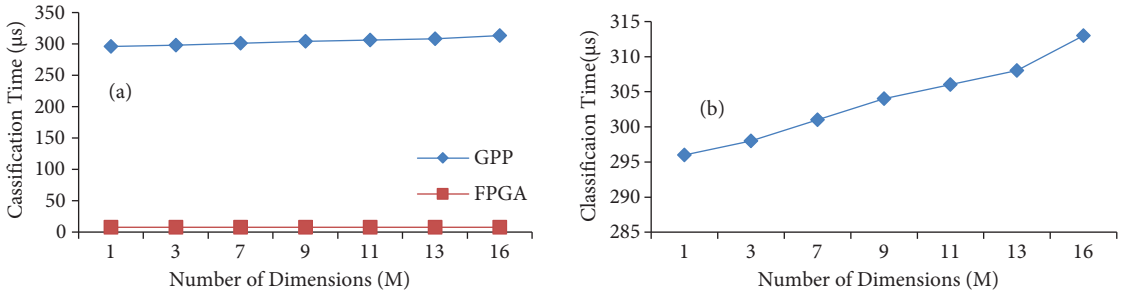

Figure 7. The effect of increasing the dimensions (M) on classification time for the FPGA and GPP implementations: (a) GPP and FPGA, (b) Enlarged GPP.

In summary, it can be stated that the FPGA generally scales better than the GPP when the dimensions of the SVs are large, mainly due to the extensive pipelining and parallelism employed in the FPGA implementation as compared to pure sequential behavior in the GPP, and due to the abundant distributed memory in medium and high end FPGAs.

6.4. DPR implementation of single-core SVM architecture based on A1 architecture

Based on the single-core implementation described in Subsection 4, a DPR implementation was constructed using the Xilinx PlanAhead 12.2 tool based on RMs having the following parameters: SVs = 20, M = 1024, and B = 8. Other RMs were also successfully created reflecting variable SVs and M. The configuration was run and verified, and full and partial bit streams were generated. The targeted Xilinx device was the XC4VSX35 FPGA. The full and partial bit streams were 1673 kB and 199 kB in size, respectively. The full and partial reconfiguration times were computed using Eq. (10):

$$\text{Configuration Time} = \frac{\text{Size of the bitstream file}}{\text{Bandwidth of the Configuration Mode}} \tag{10}$$

The results reported here are based on the JTAG as configuration mode having a bandwidth of 66 Mbps. Accordingly, the full reconfiguration time was found to be 202.78 ms while the partial reconfiguration time was 24.12 ms, leading to a speed-up in partial reconfiguration time of $\sim 8\times$ compared to a full reconfiguration of the FPGA.

This means that partially reconfiguring the FPGA not only leaves other tasks running on the FPGA uninterrupted, but also provides quick reconfiguration.

On the other hand, the SVM DPR implementation was inferior to the equivalent non-DPR implementation in terms of clock speed. The DPR implementation achieved 94.8 MHz while the non-DPR implementation achieved 137 MHz, leading to a drop in clock frequency by 31% when using DPR. This drop in clock speed could be attributed to the routing algorithm employed by the tool, particularly related to routing the DSP48 blocks with respect to other logical resources.

As for the effect of using DPR on the area footprint for the aforementioned SVM classifier, it can be stated that DPR had negligible effects, where the DPR utilized 1908 CLB slices compared to 1941 for the non-DPR implementation, leading to 12% utilization in CLB slices in both implementations for the specific used FPGA.

In addition, the second implementation, which was based on having a reconfigurable single-core within the quad-core implementation of the SVM classifier, was configured with the same parameters used for the aforementioned implementation. As such, the size of the RP was identical to the aforementioned single-core DPR implementation, leading to a partial reconfiguration time of 24.12 ms and speed-up of $\sim 8\times$ in reconfiguration time. Figure 8 shows the floorplan of this DPR implementation compared with the non-DPR quad core implementation. The two implementations are identical in partial reconfiguration speed-up since they are both based on the same FPGA and the RPs are identical in size, having the same parameters of SVs = 20, M = 1024, and B = 8. However, the second implementation illustrates an additional advantage of being able to dynamically reconfigure a single SVM core while leaving the other three SVM cores uninterrupted as well as any other tasks running on the same FPGA.

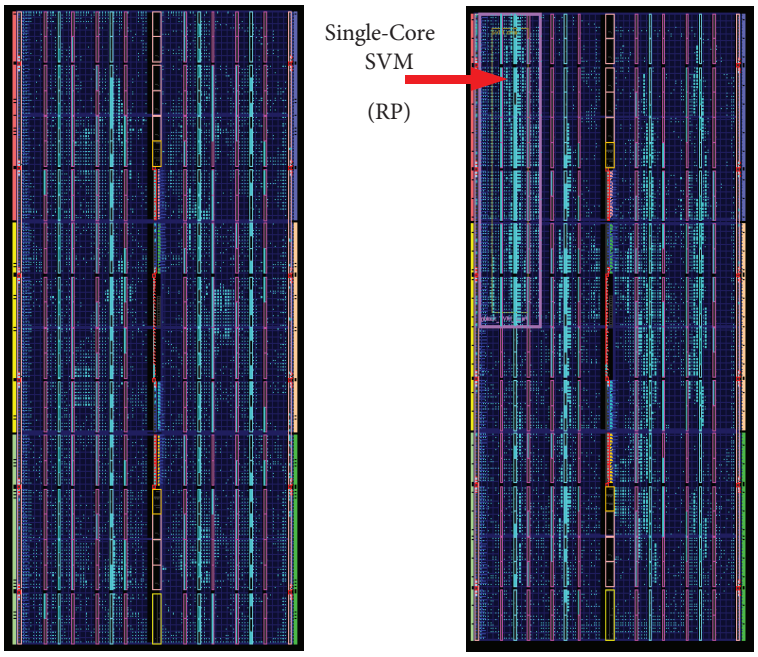

Figure 8. The floorplan image of: (a) the non-DPR implementation of the quad-core SVM classifier, (b) a DPR implementation based on a reconfigurable single-core.

The reconfiguration times obtained above using Eq. (10) are estimations of the reconfiguration times. However, in a previous work reported in [22] on the DPR implementation of K-means clustering, it was demonstrated that the difference between actual measurement and estimated measurement using Eq. (10)

was small and thus can be negligible. As such, Eq. (10) provides valid estimations of the reconfiguration time [22].

6.5. DPR implementation of multicore SVM architecture based on A1 architecture

When all four SVM cores within the quad-core classifier were made reconfigurable, the full bit stream was 1673 kB in size, while the partial bit stream was 199 KB for each of the four cores. Consequently, the full and partial reconfiguration times were 202.78 ms and 24.12 ms, respectively, resulting in a speed-up in partial reconfiguration time of $\sim 8\times$ over full chip reconfiguration. Figure 9 illustrates the floorplan of the implementation, highlighting the four RPs and the area footprint occupied by each core targeting the Xilinx XC4VSX35 FPGA.

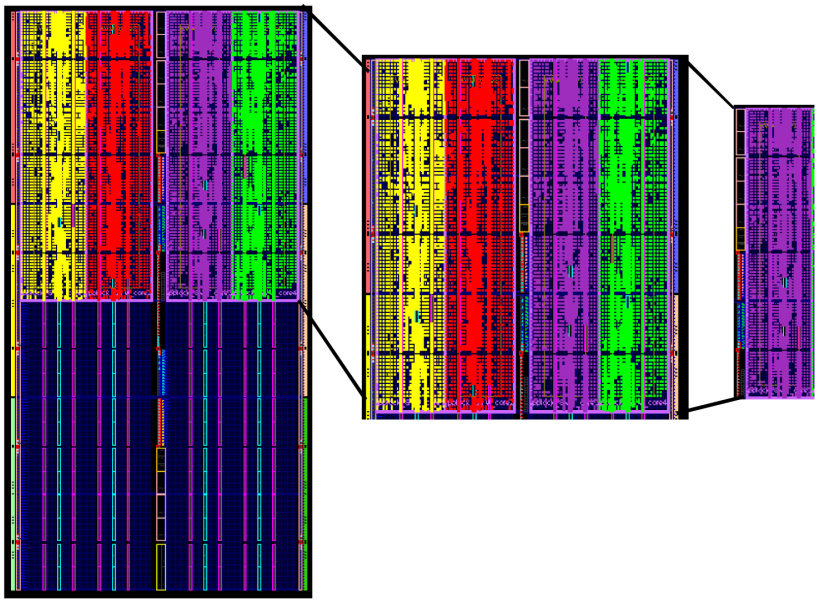

Figure 9. The floorplan image of the DPR implementation of the quad-core SVM classifier illustrating the area footprint of the quad core as well as the single core.

7. Summary and conclusion

In this work, five novel FPGA implementations of the SVM classifier were presented. At first the detailed architectures of two FPGA implementations of the SVM classifier were given. These two architectures were called A1 and A2, respectively, which were based on a linear systolic array of PEs to partially compute the linear kernel of the classification decision function. The latter is the most computationally demanding part, and one that is a candidate for hardware acceleration due to its inherent parallelism. The number of PEs in each architecture are different, whereby the number of PEs in the first architecture is equivalent to the number of SVs, and it is equivalent to the number of dimensions (M) in the second architecture. When comparing the performance of the two FPGA implementations with equivalent implementations running on the GPP, A1 achieved speed-up of $\sim 61\times$ while A2 achieved $\sim 49\times$.

Secondly, three novel DPR implementations of the SVM classifier were presented based on having single or multicore SVM classifiers dynamically reconfigurable. All of these proposed DPR architectures attained speed-up of $\sim 8\times$ in reconfiguration time over full device reconfiguration.

The multicore DPR implementation of the SVM classifier benefits from added flexibility in altering the contents of each core during run-time without affecting the operation of other tasks, speed-up in reconfiguration time, and being relocatable. This implementation caters to the requirements of server solutions where cores can be added, modified, moved around the FPGA, or removed according to the user's requests. In addition, this multicore SVM classifier can form the basic block for ensemble applications of the SVM classifier.

FPGAs have an additional advantage of being lower power devices compared to GPPs; as a result, FPGAs consume lower energy than GPPs. On the other hand, the cost of purchasing FPGAs is generally more expensive than GPPs given that the two devices are from the same generation or have comparable resources [22].

In conclusion, the hardware implementation of the SVM classifier on FPGAs realizes high performance customized solutions applied to microarray research and outperforms GPPs in terms of execution and energy consumption.

Future work will include implementing partially reconfigurable SVM training on FPGAs and an ensemble SVM classifier. Furthermore, future goals will include testing the SVM cores with benchmark datasets instead of synthetic data and running the DPR implementations on boards housing state-of-the-art FPGAs such as Virtex-7 that can accommodate data of high dimensions and more logic resources. Additionally, the possibility of incorporating a time-multiplexed reconfigurable on-chip training core will be investigated.

Acknowledgment

This work was supported by the Public Authority of Applied Education and Training (PAAET) in Kuwait to sponsor the PhD study of Hanaa Hussain at Edinburgh University.

References

[1] Stekel D. Microarray Bioinformatics. 1st ed. Cambridge, UK: Cambridge University Press, 2003.

[2] Cristianini N, Shawe-Taylor J. An Introduction to Support Vector Machines and Other Kernel-Based Learning Methods. 1st ed. Cambridge, MA, USA: Cambridge University Press, 2000.

[3] Mukherjee S. Classifying microarray data using support vector machines. In: Berrar DP, Dubilzky W, Granzow M, editors. A Practical Approach to Microarray Data Analysis. 1st ed. New York, NY, USA: Springer, 2009. pp. 1-19.

[4] Cho S, Won H. Machine learning in DNA microarray analysis for cancer classification. In: Proceedings of The First Asia-Pacific Conference on Bioinformatics; 4–7 February 2003; Seoul, Korea. pp. 189-198.

[5] Vapnik V. The Nature of Statistical Learning Theory. 2nd ed. New York, NY, USA: Springer-Verlag, 2000.

[6] Brown MP, Grundy WN, Lin D, Cristianini N, Sugnet CW, Furey TS, Ares M Jr, Haussler D. Knowledge-based analysis of microarray gene expression data using support vector machines. P Natl Acad Sci USA 2000; 97: 262-267.

[7] Anguita D. A digital architecture for support vector machines: theory, algorithm, and FPGA implementation. IEEE T Neural Networ 2003; 12: 993-1009.

[8] Anguita D, Carlino L, Ghio A, Ridella S. A FPGA core generator for embedded classification systems. J Circuit Syst Comp 2011; 20: 263-282.

[9] Gomes Filho J, Raffo M, Strum M, Jiangg Chau W. A general-purpose dynamically reconfigurable SVM. In: Proceedings of the 6th Southern Programmable Logic Conference; 24-26 March 2010; Ipojuca, Brazil. pp. 107-112.

[10] Woo Wee J, Ho Lee C. Concurrent support vector machine processor for disease diagnosis. In: Pal NR, Kasabov N, Mudi RK, Pal S, Parui SK, editors. ICONIP. Berlin, Germany: Springer-Verlag, 2004; pp. 1129-1134.

[11] Pina-Ramirez O, Valdes-Cristerna R, Yanez-Suarez O. An FPGA implementation of linear kernel support vector machines. In: Proceedings of the International Conference on Reconfigurable Computing and FPGAs; 27–29 September 2006; San Luis Potosi, Mexico. pp. 1-6.

[12] Papadonikolakis M, Bouganis CS. Efficient mapping of Gilbert's algorithms for SVM training on large-scale classification problems. In: Proceedings of the International Conference on Field Programmable Logic and Application; 8–10 September 2008; Heidelberg, Germany. pp. 385-390.

[13] Cadambi S, Durdanovic I, Jakkula V, Sankaradass M, Cosatto E, Chakradhar S, Grad HG. A massively parallel FPGA-based coprocessor for support vector machines. In: Proceedings of the 17th IEEE Symposium on Field Programmable Custom Computing Machines; 5–9 April; Napa, CA, USA. pp. 115-122.

[14] Kyrkou C, Theocharides T. SCoPE: Towards a systolic array for SVM object detection. IEEE Embed Syst Lett 2009; 1: 46-49.

[15] Papadonikolakis M, Bouganis C. A novel FPGA-based SVM classifier. In: Proceedings of the International Conference on Field-Programmable Technology; 8–10 December 2010; Beijing, China. pp. 283-286.

[16] Kim S, Lee S, Min K, Cho K. Design of support vector machine circuit for real-time classification. In: Proceedings of the 13th International Symposium on Integrated Circuits; 12–14 December 2011; Singapore. pp. 384-387.

[17] Papadonikolakis M, Bouganis C. Novel cascade FPGA accelerator for support vector machines classification. IEEE T Neural Networ Learn Syst 2012; 23: 1040-1052.

[18] Patil RA, Gupta G, Sahula V, Mandal AS. Power aware hardware prototyping of multiclass SVM classifier through reconfiguration. In: Proceedings of the 25th International Conference on VLSI Design; 7–11 January 2012; Hyderabad, India. pp. 62-67.

[19] Chang C, Lin C. LIBSVM: A library for support vector machines. ACM T Intell Syst Technol 2011; 2: 1-27.

[20] Hussain H, Benkrid K, Seker H. Dynamic partial reconfiguration based implementation of SVM classifier for classifying microarray data. In: The 35th Annual International Conference of IEEE Engineering in Medicine and Biology Society; 3–7 July 2013; Osaka, Japan. pp. 3058-3061.

[21] Xilinx Corp. Xilinx Partial Reconfiguration Guide ug702. San Jose, CA, USA: Xilinx Inc., 2010.

[22] Hussain H, Benkrid K, Ebrahim A, Erdogan A, Seker H. Novel dynamic partial reconfiguration implementation of K-means clustering on FPGAs: comparative results with GPPs and GPUs. Int J Reconfig Comput 2012; 2012: 135926.