A Classification Processor for a Support Vector Machine with embedded DSP slices and block RAMs in the FPGA

Yuki Ago, Koji Nakano, Yasuaki Ito
Department of Information Engineering
Hiroshima University
Kagamiyama 1-4-1, Higashi-Hiroshima, 739-8527, JAPAN

***Abstract*—This paper presents an FPGA implementation of a Support Vector Machine (SVM) classification using the DSP slices and block RAMs in the Xilinx Virtex-6 family FPGA. In our approach, the SVM classification is performed by the multiple DSPs. Our implementation supports 3 types of kernel functions; the sigmoid kernel, the polynomial kernel, and the RBF kernel. We connect DSPs with the built-in cascade logic in a DSP slice. Thus, our architecture consists of a cascaded DSP pipeline and process the input data with this pipeline. The number of DSP slices included in this cascade connection is equal to the number of the support vectors in the SVM. We have implemented the processor core which includes 768 DSPs for SVM classification in a Xilinx Virtex-6 FPGA XC6VLX240T-FF1156. The implementation results show that it can be implemented in the FPGA with 768 DSP48E1 slices, 800 block RAMs and 17680 slices. It runs in 370.096MHz clock frequency and can evaluate the SVM classification for 128-dimensional feature space data 2.89×10^6 times per second.**

***Keywords*-Support vector machine, Classification, FPGA, Embedded DSP slices, Embedded block RAMs**

I. Introduction

Recently, there are many researches that apply machine learning algorithms such as Support Vector Machine (SVMs) and Multi Layer Perceptron (MLP) to the pattern recognition, signal processing or image processing. These algorithms train its network with supervised machine learning before accepting the unlearned input data. The classifier achieved by the machine learning is widely used for pattern recognition. The SVM is one of the machine learning algorithms that train a classifier for a two-class classification. It is well known that SVMs often provide better classification performance than other machine learning algorithms on reasonably sized datasets [1]. The reason why SVMs achieve better generalization ability is that its learning algorithm employs two advanced methods called *margin maximization* and *kernel trick*.

A Field Programmable Gate Array (FPGA) is a programmable logic device designed to be configured by the customer or designer by hardware description language after manufacturing. The most common FPGA architecture consists of an array of logic blocks, I/O pads, DSP slices, block RAMs and routing channels. Furthermore, recent FPGAs have embedded DSP slices and block RAMs that make a higher performance and a broader application. Since FPGA chips maintain relative lower price and its programmable features, it is widely used in those fields which need to update architecture or functions frequently such as communication and education areas. They are widely used in consumer and industrial products for accelerating processor intensive algorithms [2], [3], [4], [5]. For the implementation of SVMs, an FPGA is a crucial hardware platform, which offers high performance and possibility to modify and change algorithms dynamically.

The Xilinx Virtex-6 series FPGAs have DSP48E1 slices equipped with a multiplier, adders, logic operators, etc [6]. More specifically, the DSP48E1 slice has a two-input multiplier followed by multiplexers and a three-input adder/subtractor/accumulator. The DSP48E1 multiplier can perform multiplication of an 18-bit and a 25-bit two's complement numbers and produce one 48-bit two's complement production. Programmable pipelining of input operands, intermediate products, and accumulator outputs enhances throughput and improves the frequency. The DSP48E1 also has pipeline registers between operators to reduce the delay. Also, the Virtex-6 FPGA XC6VLX240T has 768 DSP48E1 slices arranged in 8 columns of 96 adjacent DSP48E1 slices. Neighboring DSP48E1 slices are connected directly through pipeline registers.

The block RAM in the Virtex-6 FPGA is an embedded memory supporting synchronized read and write operations. In Virtex-6 FPGAs, it can be configured as a 36k-bit dual-port RAMs, FIFOs, or two 18k-bit dual-port RAMs. In our architecture, it is used as two 1k$\times$18-bit dual-port RAMs.

In this paper, we propose an FPGA implementation for the SVM classification using DSP slices and block RAMs effectively. The SVM classification is an SVM that has been trained and classifies unlearned data. We present the architectures that perform the computation of an SVM classification with DSP slices and block RAMs. Our new idea includes: **(i) Efficient Usage of DSP slices and Block RAMs:** DSP slices basically perform multiplication and accumulation to compute the dot-product and the squared Euclidean distance in the kernel function of the SVM classification. DSP slices are directly connected and work in pipeline fashion. On the other hand, block RAMs are used to store parameters obtained in the training phase of the SVMs and used as a look-up-table to compute exponential functions in the kernel function. **(ii) Fully Pipelined Architecture:** We design our new SVM architecture as a fully pipelined one using Virtex-6 FPGA XC6VLX240T that has 768 DSP48E1 slices arranged in 8 columns of 96 adjacent DSP48E1 slices. Neighboring DSP48E1 slices are connected directly through pipeline registers. Considering the structure, we design the circuit of the SVM classification efficiently using DSP slices.

Using these ideas, our new architecture for the SVM classification with 128-dimension feature space, 760 support vectors, uses 768 DSP slices and 800 block RAMs. The computing time necessary to classify an input feature vector is 338 clock cycles. Since the circuit works in fully pipelined architecture, after an input data is input, the next input data can be input immediately. The experimental results shows that our circuit runs in 370.096MHz and can evaluate 2.89×10^6 times per second for the SVM classification with 128-dimension feature space. Note that in this paper, our goal is to accelerate the classification of the SVM that has already been trained in advance, not to accelerate the training of the SVM. In other words, the parameters of the SVM obtained by training phase are computed on a host PC. These parameters are stored in block RAMs of the FPGA connected to a host PC.

Many hardware algorithms with FPGAs to accelerate the computation of the SVM classification have been proposed in past [7], [8], [9], [10], [11], [12]. As far as we know, however, there is no previously published work that fully utilizes DSP slices and block RAMs for the SVM classification.

This paper is organized as follows. Section II introduces the SVM. We show the FPGA architecture for the SVM classification in Section III. Section IV evaluates the performance of our implementation. We show the experimental results in Section V. Finally, Section VI concludes the paper.

II. Support Vector Machine

The main purpose of this section is to review the SVM. The SVM is one of the supervised machine learning algorithms that train a classifier for a binary classification [13]. The SVM classifies an input data set into two classes that are labeled as $+1$ or -1. The SVM constructs separating hyperplanes between them. The separating hyperplanes that best separates the two classes is called the maximum-margin hyperplane and forms the decision boundary for classification. The data sets lying at the boundary for each class are called *support vectors*. The SVs are obtained in the training phase and are utilized to classify unknown data. Also, when two data classes are not linearly separable, a kernel function is used to project data to a higher dimensional space, which is called *kernel trick*.

Suppose that we have a data set $\mathbf{T}$ of l samples $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_l$ in a d-dimensional space and these samples exactly belong to one of the two classes $(+1, -1)$. The data set $\mathbf{T}$ is represented as $\mathbf{T} = \{(\mathbf{x}_i, y_i) | i \in \{1, \ldots, l\}, \mathbf{x}_i \in \Re^d, y_i \in \{+1, -1\}\}$. The training an SVM is equivalent to solving the following quadratic optimization problem:

$$\begin{aligned} \text{minimize} \quad & \frac{1}{2} \sum_{i=1,j=1}^{l} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) - \sum_{i=1}^{l} \alpha_i (1) \\ \text{subject to} \quad & 0 \le \alpha_i \le C \quad \text{for } i = 1, \ldots, l, \\ & \sum_{i=1}^{n} y_i \alpha_i = 0, \end{aligned}$$

where $\mathbf{x}_i$ are the support vectors, α_i are the Lagrangian coefficients, and $K(\cdot, \cdot)$ is the kernel function. The classification function $f(\mathbf{x})$ is

$$f(\mathbf{x}) = \text{sign}(\sum_{i=1}^{n} \alpha_i y_i K(\mathbf{x_i}, \mathbf{x}) + b), \tag{2}$$

where $\mathbf{x}$ is the new feature vector and b is the threshold, $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ are the support vectors, and $\alpha_1, \alpha_2, \ldots, \alpha_n$, and b are the parameters. During the training phase, $\mathbf{x_i}$, α_i and b are obtained. In our proposed architecture, this equation is computed for given input feature vectors. Regarding the kernel function, there are three different types of kernel basis functions that are commonly used; *the sigmoid kernel*, *the polynomial kernel*, and *the radial basis function (RBF) kernel*. These are used to create nonlinear classifiers by applying the kernel trick [14]. The definition of the kernels is as follows; the sigmoid kernel: $K(\mathbf{u}, \mathbf{v}) = \tanh(s\mathbf{u} \cdot \mathbf{v} - t)$ for $s, t > 0$, the polynomial kernel: $K(\mathbf{u}, \mathbf{v}) = (\mathbf{u} \cdot \mathbf{v} + 1)^d$, and the RBF kernel: $K(\mathbf{u}, \mathbf{v}) = \exp(-\frac{\|\mathbf{u}-\mathbf{v}\|^2}{2\sigma^2})$ for $\sigma > 0$. The parameters s, t, d, and σ are determined in the training phase. Also, $\mathbf{u} \cdot \mathbf{v}$ and $\|\mathbf{u} - \mathbf{v}\|^2$ represent the dot-product and the squared Euclidean distance between two vectors $\mathbf{u}$ and $\mathbf{v}$, respectively. In fact, an SVM with the sigmoid kernel function is equivalent to a three-layer-perceptron neural network [15].

In our work, the training phase is performed on a host PC to determine appropriate parameters. There parameters are stored in block RAMs of the FPGA connected to a host PC. Therefore, to classify a given feature vector $\mathbf{x}$, our proposed circuit evaluates the classification function $f(\mathbf{x})$ in Eq. (2) with one of the above kernel functions using the parameters determined in the training phase.

III. The Architecture of an SVM Classification Circuit

This section describes the architecture of our SVM classification circuit. The main idea of our architecture is to effectively utilize the cascaded DSP48E1 slices and block RAMs in Xilinx Virtex-6 FPGA. We use Xilinx Virtex-6 family FPGA XC6VLX240T-FF1156 as the target device [6]. It consists of columns of slices each of which includes two Configurable Logic Blocks (CLBs), programmable Input/Output Blocks (IOBs), DSP48 slices, and 36k-bit dual-port block RAMs.

The classification function of the SVM in Eq. (2) is divided into two steps of computations. The first step is to compute the kernel function $K(\cdot,\cdot)$. As shown in the above, there are three typical kernel functions; the sigmoid kernel, the polynomial kernel, and the RBF kernel. We present the architecture that supports these three kernel functions. The second step is to compute the sum of the products of the kernel function and the weights $\alpha_i y_i$. In order to implement the computation, it is necessary to compute the dot-product between support vectors and an input feature vector. In our architecture, we use DSP slices to compute the dot-product. To store the weights $\alpha_i y_i$, we use block RAMs. Also, we use block RAMs to store a table to compute the exponential function in the kernel function.

A. Data representation

The choice of data precision is guided by the implementation cost in terms of area, simplicity of design, speed and power consumption. On the one hand, higher precision will lead to less quantization error in the final implementation. On the other hand, lower precision will produce more compaction and faster designs with less power consumption. A trade-off choice needs to be made depending on the given application and available FPGA resources.

In our work, in order to minimize chip space and computation time, short fixed-point representations of numbers are used. The data format (input, weight, intermediate result and output) is 18-bit fixed point number in our system, which consists of 1-bit sign, 3-bit integer, and 14-bit fraction based on two's complement. According to papers [16], [17], [18], the precision is sufficient to perform the computation in the SVM. The data format is just like *SIII.FFFFFFFFFFFFFF*, where *S* is sign bit, *I* is integer bit and *F* is fraction bit. Thus, the discrete error is at most $\epsilon = 2^{-14} \approx 6.1 \times 10^{-5}$, and the maximum is $0111.11111111111111 = 8 - \epsilon$ and the minimum is $1000.00000000000000 = -8$. Consequently, real numbers in our system are in the rage $[-8, 8-\epsilon]$ with precision 6.1×10^{-5}. Also, if the interim value is out of this range, it is rounded either to the maximum or the minimum.

B. The usage of block RAMs

In this section, we describes how block RAMs are used. We have used three types of block memories, as follows. **(i) SV-RAM:** Block RAMs are used to store the feature values of each support vector $\mathbf{x_i}$. The number of SV-RAMs corresponds to the number of support vectors determined in the training phase. To compute the kernel function, the stored data is used. **(ii) K-RAM:** Block RAMs are used to store a table to compute the exponential function in the kernel function. **(iii) W-RAM:** Block RAMs are used to store the weight values $\alpha_i y_i$ and the offset value b in Eq. (2).

C. Implementation of the kernel functions

One of the design challenges with the SVM on the FPGA is the kernel function. There are three typical kernel functions; the sigmoid kernel, the polynomial kernel and the RBF kernel as shown in Section II. In our work, we present the architecture supporting these three kernel functions.

According to the kernel functions shown in Section II, the computation of them mainly consists of two parts; (1) the dot-product or the squared Euclidean distance between a support vector and an input feature vector, and (2) the exponential function. The dot-product is computed in the sigmoid kernel and the polynomial kernel, and the squared Euclidean distance is computed in the RBF kernel. To compute the dot-product and the squared Euclidean distance, we use one DSP slice for each. On the other hand, a K-RAM is used to compute the exponential function. In the followings, we explain the details of the implementation.

Figure 1 illustrates our circuit for the computation of the dot-product with one DSP slice and one block RAM. Given the elements of an input feature vector $\mathbf{x}$ one by one, the corresponding elements of the support vector $\mathbf{x}_i$ stored in the SV-RAM are input to the DSP slice and the DSP slice computes the dot-product between two vectors. The configuration of the DSP slice in this circuit consists of an 18×18 bit two's complement multiplier, a 48-bit adder, and a 48-bit register. The DSP slice computes the dot-product with them. After the computation of the dot-product, the result is output. Note that this circuit is used for one support vector and an input feature vector. Therefore, if the number of the support vectors in the SVM is n, n circuits are necessary.

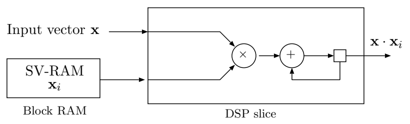


Figure 1. The circuit for the computation of the dot-product in the sigmoid and polynomial kernels

On the other hand, Figure 2 illustrates our circuit for the computation of the squared Euclidean distance with one DSP slice and one block RAM. The structure of the circuit is similar to that for the above. The main difference is that an additional subtractor with CLBs is used. The subtractor computes the difference of the corresponding elements of an input vector and a support vector. The DSP slice computes the sum of squares. Similarly, if the number of the support vectors in the SVM is n, n circuits are necessary.

In order to compute the exponential function in the kernel functions, we take a look-up-table implementation using a block RAM. For each kernel function, the look-up-table is used, as follows.

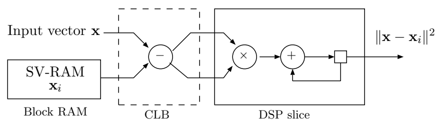


Figure 2. The circuit for the computation of the squared Euclidean distance in the RBF kernel

The sigmoid kernel: The value of $\tanh(sp-t)$ is stored in the address of p.

The polynomial kernel: The value of $(p+1)^d$ is stored in the address of p.

The RBF kernel: The value of $\exp(-\frac{p}{2\sigma^2})$ is stored in the address of p.

Given the results of the above circuit, each exponential function are computed with a K-RAM as shown in Figure 3. Recall that the data format is 18-bit fixed point number in our system. If we implement full 18-bit precision, for computing the above, we need a look-up-table of size 18×2^{18} bits. However, the size of a block RAM is 18×2^{10}. Thus, we use for 18kbit block RAMs and the most significant 12 bits of interim values of the result of the dot-products and the squared Euclidean distance as the address of the look-up-table.

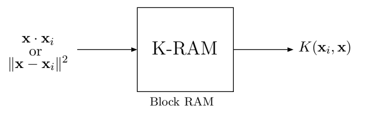

$\mathbf{x}\cdot\mathbf{x}_i$ or $\|\mathbf{x}-\mathbf{x}_i\|^2$
K-RAM
$K(\mathbf{x}_i,\mathbf{x})$
Block RAM

Figure 3. The circuit for the exponential function using a K-RAM

D. Implementation of the SVM classification

After the computation of the kernel function, to complete the computation of $f(\mathbf{x})$ in Eq. (2), the computation of the n dot-products between $K(\mathbf{x}_i,\mathbf{x})$ and $\alpha_i y_i$ for $(1 \leq i \leq n)$, and the comparison with b are necessary. Figure 4 illustrates our circuit for the computation. The circuit for this dot-product is very similar to that for the dot-product in the kernel function. The difference is using a W-RAM storing the coefficients $\alpha_1 y_1, \alpha_2 y_2, \ldots, \alpha_n y_n$ instead of an SV-RAM. For the result of the dot-product, it is compared with a threshold value b. If the result is greater than or equal to b, the comparator outputs 1 showing that the input feature is labeled as $+1$. If not, the comparator outputs 0 showing that the input feature is labeled as -1.

E. The fully pipelined architecture using cascaded DSP slices

In this section, we show a fully pipelined architecture for the SVM classification using cascaded DSP slices using

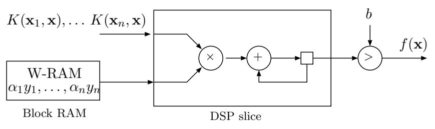

$K(\mathbf{x}_1,\mathbf{x}),\ldots K(\mathbf{x}_n,\mathbf{x})$
W-RAM $\alpha_1 y_1,\ldots,\alpha_n y_n$
Block RAM
DSP slice
b
$f(\mathbf{x})$

Figure 4. The circuit for the classification

the above circuits. The outline of the whole circuit of the SVM classification is illustrated in Figure 5. Recall that the DSP48 slice is a configurable slice with a multiplier, an adder, and registers. In our architecture, we use three types of configurations illustrated in Figure 6. The readers can find that the configuration of the DSP slice has been shown in the above. The difference between them is that Type 2 employs two additional registers. Since ports $BCIN$ and $BCOUT$ in adjacent DSP48 slices are directly connected, one Type 1 DSP48 slice and several Type 2 DSP48 slices can be connected as illustrated in Figure 5. Note that the computation for the RBF kernel cannot be performed within only one DSP slice as mentioned the above. The additional subtractor and the two registers are implemented by CLB slices. Also, Type 3 DSP48 slice is utilized in the multiple cascaded DSP slices. According to this architecture, the circuit can work in the fully pipelined architecture.

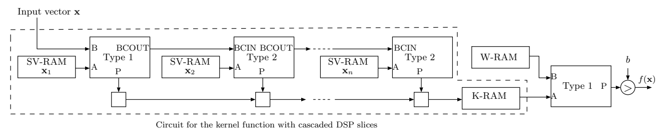
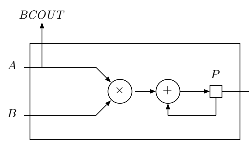
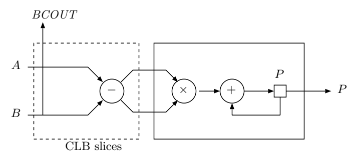


Figure 5. Outline of the SVM classification circuit with the cascaded DSP slices in a single column

However, there is a limitation for the number of DSP slices that can be directly connected. Also, the connection is used only in the adjacent DSP slices, that is, the ports $BCIN$ and $BCOUT$ are connected only to $BCOUT$ and $BCIN$ in the adjacent DSP slice, respectively. For example, the limitation of the Virtex-6 FPGA XC6VLX204T, which is the target device in our wok, is 96 DSP slices. Therefore, we cannot construct more than 96 DSP slices as illustrated in Figure 5. Since the number of the cascaded DPS slices corresponds to the number of support vectors, our architecture accept an SVM with at most 96 support vectors. In our work, to support the SVM with more support vectors, a single column of the cascaded DSP slices are divided into multiple columns as illustrated in Figure 7. In the circuit, to compute the dot-product between the outputs of a K-RAM and a W-RAM with Type 1 DSP slice in Figure 5, we use m Type 3 DSP slices and m W-RAMs, where m is the number of the columns of the cascaded DSP slices. Type 3 DSP slices are used to compute the local dot-product in the corresponding column and to sum them by selecting

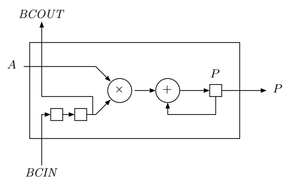


(a) The sigmoid and polynomial kernels

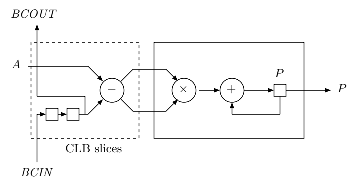


(b) The RBF kernel

Type 1: multiplier accumulator

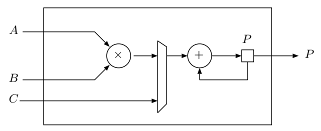


(a) The sigmoid and polynomial kernels



(b) The RBF kernel

Type 2: multiplier accumulator with two registers



Type 3: multiplier accumulator with an additional input port

Figure 6. Three types of configurations of the DSP48 slice

the function. Using this circuit, we can accept an SVM with more support vectors. Note that since the W-RAM in the single column architecture is also divided into m parts, the number of entries of each W-RAM is reduced to $\frac{1}{m}$. However, the number of input/output ports is limited, the number of W-RAMs increases from 1 to m.

IV. Performance Evaluation

Let us evaluate the performance of the architecture of the SVM classification. Again, let d, n, and m denote the numbers of dimensions of the feature space, support vectors, and columns of the cascaded DSPs. Recall that our circuit can work in fully pipelined architecture. Our circuit works as follows: The input feature vector of d elements to be classified is provided to the circuit such that each element is given to the circuit in every clock cycle one by one. Hence, d elements of the feature vector are provided in d clock cycles. Therefore, the circuit takes $d+2\lceil\frac{n}{m}\rceil+m+12$ clock cycles to classify an input feature vector. After inputting the feature vector, the next input feature vector can be provided immediately.

Next, let us evaluate the utilized resources of the circuit. From Figure 7, the number of the DSP slices is $m+n$. Also, since the number of entries of the 18k-bit block RAM is 1024, the numbers of 18k-bit block RAMs for each of an SV-RAM, a K-RAM, and a W-RAM are $n\lceil\frac{d}{1024}\rceil$, 4m, and

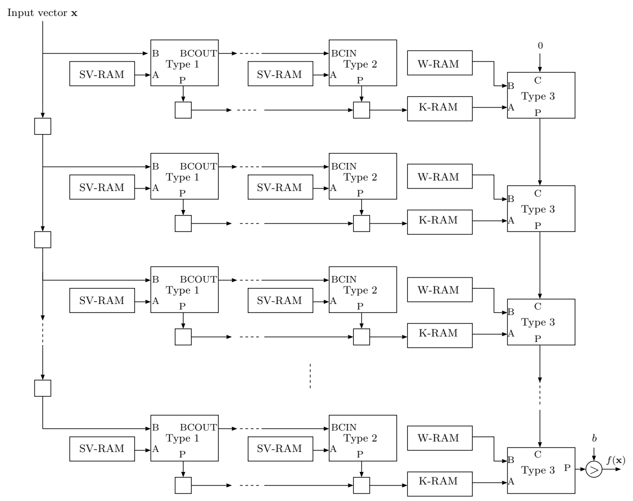


Figure 7. The architecture with the cascaded DSP slices divided into the multiple columns

$m\lceil\frac{d}{1024m}\rceil$, respectively.

According to the above evaluation, for example, if we have implemented a circuit of an SVM classifier with 128-dimension feature space, 760 support vectors, and 8 columns of cascaded DSP slices, the computing time for evaluating an input feature vector is $128+2\times\lceil\frac{760}{8}\rceil+8+12=2\times 95+8+12=338$ clock cycles. Input feature vectors can be provided to the circuit in every 128 clock cycles. Also, the number of utilized DSP slices is $760+8=768$. The number of 18k-bit block RAMs is $760\lceil\frac{128}{1024}\rceil+4\times 8+8\lceil\frac{128}{1024\times 8}\rceil=760\times 1+4\times 8+8\times 1=800$.

V. Experimental results

We have evaluated the performance of our SVM classifier circuit using the Xilinx Virtex-6 family FPGA 6VLX240T-FF1156. Table I shows summarizes the experimental results for three types of the SVM classifiers using ISE Foundation 14.1. The numbers of used DSP48 slices and 18k-bit block RAMs, and clock cycles necessary to classify one input feature vector are equal to the results of the evaluation presented in Section IV. We note that the number of CLB slices for the sigmoid and polynomial kernels and the RBF kernel are different because the circuit of the kernel function is different shown in Section III. Therefore, since in the circuit for the RBF kernel, additional CLB slices are necessary, the number of used CLB slices for the RBF kernel in is more than that for the sigmoid and polynomial kernels. However, the clock frequency of them does not change since pipeline registers are inserted to both circuits. According to the table, the clock frequency does not depend on the numbers of support vectors and columns of cascaded DSP slices. The throughput corresponds to the number of input vectors that can be classified in one second and each throughput is

Table I
PERFORMANCE EVALUATION FOR THREE TYPES OF SVM CLASSIFICATION CIRCUITS

	# of dimensions of feature space	128	128	128
	# of support vectors	95	475	760
	# of columns of cascaded DSP slices	1	5	8
The sigmoid kernel The polynomial kernel	DSP48E1 slices	96	480	768
	18k-bit block RAMs	100	500	800
	CLB slices	3930	19650	31440
	Clock frequency [MHz]	370.096	370.096	370.096
	Time [clock cycles]	331	335	338
	Throughput [1/s]	2.89×10^6	2.89×10^6	2.89×10^6
The RBF kernel	DSP48E1 slices	96	480	768
	18k-bit block RAMs	100	500	800
	CLB slices	7332	36660	58688
	Clock frequency [MHz]	370.096	370.096	370.096
	Time [clock cycles]	331	335	338
	Throughput [1/s]	2.89×10^6	2.89×10^6	2.89×10^6

identical. This is because the throughput depends only on the number of dimensions of the feature space.

VI. CONCLUSION

In this paper, we presented an FPGA implementation for an SVM classification efficiently using DSP slices and block RAMs and implemented it in the Xilinx Virtex-6 family FPGA. Our circuit can work in fully pipelined architecture and support three typical kernel functions; the sigmoid kernel, the polynomial kernel, and the RBF kernel. Experimental results show that our implementation can perform in extremely high speed and throughput.

REFERENCES

[1] S. Marsland, *Machine Learning: An Algorithmic Perspective*. Chapman and Hall/CRC, 2009.

[2] J. L. Bordim, Y. Ito, and K. Nakano, "Accelerating the CKY parsing using FPGAs," *IEICE Transactions on Information and Systems*, vol. E86-D, no. 5, pp. 803–810, May 2003.

[3] ——, "Instance-specific solutions to accelerate the CKY parsing for large context-free grammars," *International Journal on Foundations of Computer Science*, pp. 403–416, 2004.

[4] Y. Ito and K. Nakano, "Efficient exhaustive verification of the Collatz conjecture using DSP blocks of Xilinx FPGAs," *International Journal of Networking and Computing*, vol. 1, no. 1, pp. 19–62, 2011.

[5] Y. Ito, K. Nakano, and S. Bo, "The parallel FDFM processor core approach for CRT-based RSA decryption," *International Journal of Networking and Computing*, vol. 2, no. 1, pp. 79–96, 2012.

[6] Xilinx Inc., *Virtex-6 Family Overview*, 2010.

[7] P. J. Pingree, L. J. Scharenbroich, and T. A. Werme, "Implementing legacy-C algorithms in FPGA co-processors for performance accelerated smart payloads," in *Proc. of IEEE Aerospace Conference*, 2008, pp. 1–8.

[8] M. Ayinala, "Low-power architecture for signal processing and classification systems," Ph.D. dissertation, The University of Minnesota, August 2012.

[9] M. Papadonikolakis and C.-S. Bouganis, "Novel cascade FPGA accelerator for support vector machine classification," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 23, no. 7, pp. 1040–1052, 2012.

[10] M. Ruiz-Llata, G. Guarnizo, and M. Yébenes-Calvino, "FPGA implementation of a support vector machine for classification and regression," in *Proc. of International Joint Conference on Neural Networks*, 2010, pp. 2037–2041.

[11] D. Anguita, S. Pischiutta, S. Ridella, and D. Sterpi, "Feedforward support vector machine without multipliers," *IEEE Transactions on Neural Networks*, vol. 17, no. 5, pp. 1328–1331, 2006.

[12] S. Kim, S. Lee, and K. Cho, "Design of high-performance unified circuit for linear and non-linear SVM classifications," *Journal of Semiconductor Technology and Science*, vol. 12, no. 2, pp. 162–167, June 2012.

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

[14] D. Anguita, A. Boni, and S. Ridella, "A digital architecture for support vector machines: Theory, algorithm, and FPGA implementation," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 14, no. 5, pp. 993–1009, 2003.

[15] A. R. Omondi and J. C. Rajapakse, *FPGA Implementations of Neural Networks*. Springer, 2006.

[16] D. Anguita, A. Boni, and S. Ridella, "Learning algorithm for nonlinear support vector machines suited for digital VLSI," *Electronics Letters*, vol. 35, no. 16, pp. 1349–1350, 1999.

[17] ——, "Digital VLSI algorithms and architectures for support vector machines," *International Journal of Neural Systems*, vol. 10, no. 3, pp. 159–170, June 2000.

[18] D. Anguita, A. Ghio, S. Pischiutta, and S. Ridella, "A hardware-friendly support vector machine for embedded automotive applications," in *Proc. of International Joint Conference on Neural Networks*, 2007, pp. 1360–1364.