HLS-based dataflow hardware architecture for Support Vector Machine in FPGA

Mohammad Amir Mansoori and Mario R. Casu
Department of Electronics and Telecommunications (DET), Politecnico di Torino, Turin, Italy
{mohammadamir.mansoori, mario.casu}@polito.it

***Abstract*—Implementing fast and accurate Support Vector Machine (SVM) classifiers in embedded systems with limited compute and memory capacity and in applications with real-time constraints, such as continuous medical monitoring for anomaly detection, can be challenging and calls for low cost, low power and resource efficient hardware accelerators. In this paper, we propose a flexible FPGA-based SVM accelerator highly optimized through a dataflow architecture. Thanks to High Level Synthesis (HLS) and the dataflow method, our design is scalable and can be used for large data dimensions when there is limited on-chip memory. The hardware parallelism is adjustable and can be specified according to the available FPGA resources. The performance of different SVM kernels are evaluated in hardware. In addition, an efficient fixed-point implementation is proposed to improve the speed. We compared our design with recent SVM accelerators and achieved a minimum of $10\times$ speed-up compared to other HLS-based and $4.4\times$ compared to HDL-based designs.**

***Index Terms*—SVM, FPGA, HLS, Hardware acceleration.**

I. Introduction

Support Vector Machine (SVM) is a supervised Machine Learning (ML) model widely used in different classification problems, such as image classification, medical diagnosis, object detection, and bioinformatics [1]. For example, in microwave imaging, SVM can detect a brain stroke from the electromagnetic scattering data [2], [3]. Since it is challenging to implement SVM in real-time embedded systems, several specialized hardware architectures have been recently proposed. Among them, those based on Field Programmable Gate Arrays (FPGAs) are preferable in embedded systems due to their flexibility and lower power. In [4], a review of recent FPGA accelerators for SVM is presented.

A parallel hardware architecture for SVM using a systolic array of vector processing units to process multiple support vectors (SVs) in parallel is proposed in [5] and extended to a cascade SVM classifier for real-time object detection in [6]. The main limitation of these works is that all the coefficients and SVs are stored in on-chip memory, which means that the number of SVs is limited by the memory resources in FPGA. In [7], another SVM architecture based on a Verilog RTL description is proposed. It uses on-chip FIFOs to store all the SVs and can process no more than 20 SVs in parallel, with no parallelism on the SVs dimension or the number of features N_f: as we will see, our design can support more features and a higher clock frequency. The works in [8]–[11] also used RTL, either created manually or with Xilinx System Generator.

High Level Synthesis (HLS) enables a more efficient design space exploration than RTL manual design. In [12], an HLS design for SVM acceleration in the application of melanoma detection is proposed, which is extended for better performance in [13], [14]. Due to the assumption of local storage of SVM coefficients, this design is tested on small-scale problems with 27 features and a maximum of 346 SVs. In [15] a linear multi-class SVM is used for brain cancer detection in Hyper-Spectral Images (HSI). Due to the large dimensions of HSI datasets, the input vector could not be stored in local memory. However, due to the linear kernel used in this work, the main part of kernel computation is processed off-line and the weighted summation of all SVs is stored as a single vector, in a way that only one vector is used in on-line computations. In [16], a methodology to extract parallelism from software code is introduced for optimized annotation of C code with HLS directives, and is tested on SVM. Although different levels of hardware parallelism are explored in this work, the authors considered local storage for the inputs and did not report the time required to read and store all the inputs.

Previous SVM works usually ignored scalability, which allows the same design to be used for larger data dimensions. Usually, coefficients and SVs are stored in on-chip memory, which is a method useful for small scale problems but cannot be used for large data dimensions and/or low-cost FPGAs due to the lack of local memory. Indeed, it is possible to do the SVM computations while reading the required data from an external memory, hence increasing the overall throughput. This is the main idea on which our dataflow design is based.

Most of the previous designs only considered binary classification or simple kernel functions. Multi-class classification is a more challenging problem requiring more computations, specially if complex kernels are used. In this paper, we propose a scalable hardware accelerator for multi-class SVM classification in FPGA by using an efficient dataflow architecture designed entirely in HLS. These are our contributions:

- A specialized dataflow hardware accelerator for SVM algorithm in FPGAs that can scale to support different data dimensions while guaranteeing a high throughput.
- Support for multi-class classification and various kernels.
- Adjustable parallelism by HLS-based configurations and efficient implementation of fixed-point design.

II. SVM background

SVM algorithm for binary classification obtains a decision boundary as a hyper-plane that maximizes the margin between two classes as shown in Fig. 1. For linear classification, the

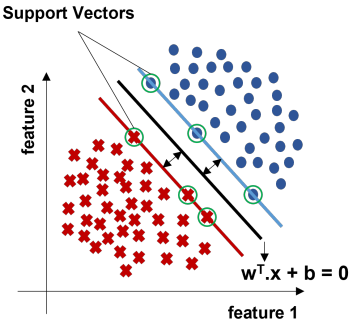


Fig. 1. SVM classification with linear kernel.

hyper-plane can be expressed as $w^T x + b = 0$, where w and b are obtained during training, and x is a 1-D input vector with N_f elements. The inputs that lie on the margin are termed *Support Vectors* (SVs). A total number of N_{SV} vectors can be stored as a 2-D array of size $N_{SV} \times N_f$. To obtain w and b, a dual problem is constructed by using Lagrange multipliers (indicated as α_i parameter for each input). After training, α_i are obtained and all the inputs for which $\alpha_i \neq 0$ are regarded as support vectors (SV_i). Finally, w and b can be obtained from α_i and SV_i and the decision boundary can be written as

$$\text{Decision} = \sum_{i=1}^{N_{SV}} \alpha_i K(x, SV_i) + b, \tag{1}$$

in which SV_i is the ith support vector and $K(\cdot)$ is one of the *Kernel* functions in Tab. I. For linear problems, the Kernel is a dot product; for non-linear problems, the Kernel will transform the input space into one where the classes are linearly separable. Note that in (1) α_i must be multiplied by the labels of the support vectors (y_i), which for clarity, we considered is done internally ($\alpha_i = \alpha_i y_i$).

TABLE I
KERNEL FUNCTIONS IN SVM.

Kernels: $K(x, SVi)$	
Linear	$x \cdot SV_i$
Radial Basis Funtion (RBF)	$exp(-\frac{\|x-SV_i\|^2)}{2\sigma^2}$
Polynomial (Poly)	$(c_1(x \cdot SV_i) + c_2)^P$
Sigmoid	$\tanh(c_1(x \cdot SV_i) + c_2)$

For binary classification, the sign of *Decision* is simply used for the prediction. For multi-class, we used *one-vs-one* (*ovo*) method, in which all pairs of classes are compared using (1) and based on the majority vote, the final prediction is computed. With N_c classes, the total number of comparisons (i.e., the decision vector size) for *ovo* is $N_d = N_c(N_c - 1)/2$.

III. PROPOSED SVM ACCELERATOR

To increase the overall throughput and reduce the on-chip storage, we propose the Dataflow hardware architecture illustrated in Fig. 2. Instead of storing all the SVM input data in on-chip memory, we read chunks of data from the external memory, transfer them through the FIFO channels, and do the subsequent computations while the next chunk is being read. We can design such a hardware architecture in HLS by defining a dataflow region between three functions in the main module related to the processing and reading the SVM data.

As shown in Fig. 2, three functions for SVM computations are *Read*, *Kernel*, and *Decision*. *Read* distributes the chunks of input read data through the FIFO channels. Since the major part of SVM computations is dedicated to the dot product between the input features, we match the size of a block of input features to a data chunk. *Kernel* reads the chunks from the FIFOs, calculates the kernel function and sends the result to the next FIFO channel. *Decision* receives the kernel output from the FIFO as well as three other input coefficients, and computes the final prediction, denoted as Vote in Fig. 2. We use Vivado HLS since we target Xilinx FPGAs and with this tool we can use a *dataflow* directive and *hls::stream* variables for the FIFO channels to obtain the implementation of Fig. 2.

A. Read SVM inputs

The data are stored as 32-bit floating-point values in an external DDR memory. Depending on the maximum DDR data width (DW_{ddr}, in bits), DDR frequency (f_{ddr}), and working clock frequency (f_w), the maximum number of 32-bit data that can be read from the memory in one clock cycle is $N = (DW_{ddr}/32) \times ((2 \times f_{ddr})/f_w)$. For large-scale problems, storing SV in FPGA memory is the main limitation. In this work, we consider SVs and the test vector (x) stored in off-chip and on-chip memory, respectively. However, with enough memory bandwidth, both of them can be stored in external memory.

The *Read* function will transfer the input data to the *Kernel* function through BF FIFO channels, as indicated in the left part of Fig. 2, in such a way that the Kernel function can process BF data in parallel. Ideally, we want $N = BF$ because it is not possible to write more than N values to the FIFO channels in each clock cycle. However, by increasing BF we speed up the Kernel function, although we slow down the Read one. This is shown in Fig. 3, where by increasing BF we increase the Initiation Interval (II)[1] of the Read function, but this is in part masked by the overlapping of Read and Kernel functions due to the pipelining effect of the Dataflow implementation, and is compensated by the higher throughput of the Kernel function due to the greater parallelism. To keep the pipeline balanced, BF and N, which are defined as configurable variables in the HLS code, should be properly related. For this, we use the SVM latency equation (T) from the maximum latency of *Read*, *Kernel*, and *Decision*

$$T \approx N_{SV} \cdot N_f \cdot \max\{\frac{II_R}{BF}, \frac{II_K}{BF}, \frac{II_D}{N_f \cdot DF}\}, \tag{2}$$

where the approximation comes from considering only the II of the various functions and from not considering the latency of the iterations. Indeed, II_R, II_K, and II_D are the II of *Read*, *Kernel*, and *Decision*, respectively. In large-scale problems, the

[1]Minimum number of clock cycles before the next input data can be received.

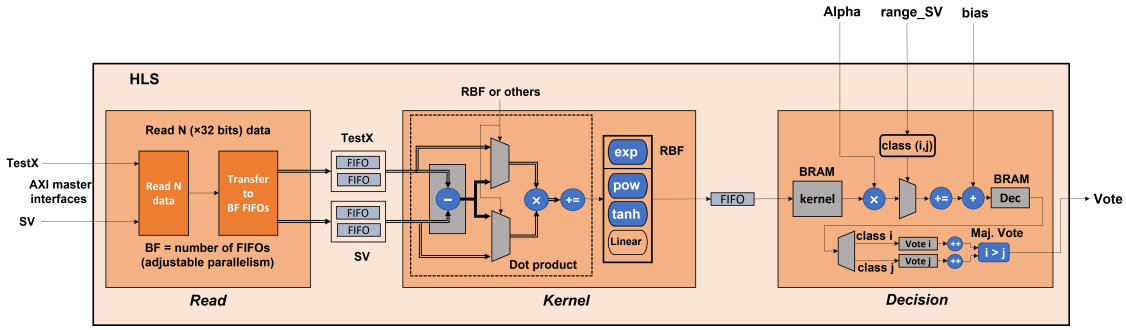


Fig. 2. Proposed SVM accelerator in HLS.

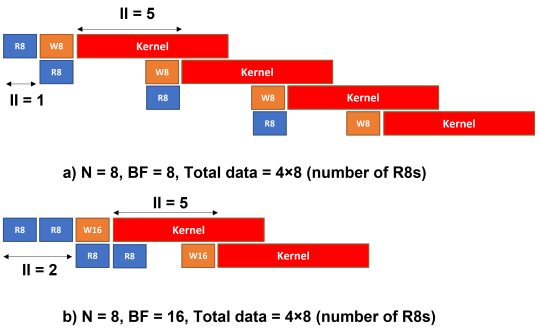


Fig. 3. Impact of the number of FIFO channels (BF) with a total of $4 \times 8 = 32$ data, (a) $BF = N$, (b) $BF = 2N$, overall latency is reduced.

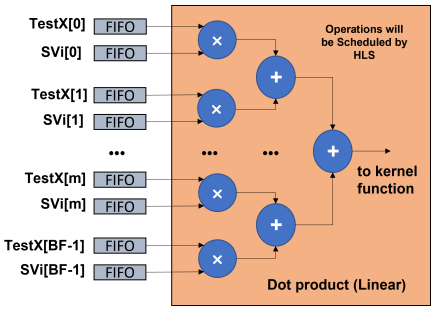


Fig. 4. Manual unrolling for kernel computation.

Decision term is small and the total latency is determined by the maximum value between II_R/BF and II_K/BF. Therefore, a balanced pipeline would require $II_R = II_K$ and since $II_R = BF/N$, we should have $II_K = BF/N$. If we have $II_K < BF/N$ we end up with a memory-bound performance, otherwise the performance would be computation-bound. In the ideal case, if II for all the functions is 1, we must select $N = BF$ to match the throughput. Note that there is an iteration latency (L) to add to the latency of each function and that *Kernel* has a higher L than *Read*. Therefore, in a balanced dataflow, the latency between *Read* and *Kernel* is determined by the *Kernel* latency. As we will see, for this reason it is sometimes preferable not to have a perfectly balanced pipeline and select, for example, $BF = 2N$: *Read* will be the dominant part, but the overall latency will decrease.

B. Kernel computation

The hardware architecture for *Kernel* computation is shown in the middle part of Fig. 2. Note that to compute the RBF kernel, we need to subtract the inputs and compute its squared norm to be used in the exponential function. For other kernels, only the dot product between the inputs are computed. This is shown in Fig. 2 by a control signal (RBF or others) for clarity.

The parallelism of the computation is matched to the number of FIFO channels, as we instantiate BF parallel elements for the dot product, with BF multipliers (and subtractors for the RBF kernel). This can be obtained by *partial unrolling* directive in HLS with a factor of BF. Due to the accumulation in the dot product, if we use one scalar variable to store the accumulation (after $+=$ operation in Fig. 2), HLS tool cannot schedule the design with partial unrolling as we expect because of the data dependency on the scalar variable. Therefore, we use manual unrolling as shown in Fig. 4 (for clarity, only linear kernel is shown) and define an array of size BF to hold the result of each multiplication. By fully partitioning this array and computing the addition of all its elements (shown as adders), the dot product will be computed.

After the dot product, a non-linear function selected based on the SVM kernel can be applied, if needed, otherwise for the linear case the dot product is passed directly to the output.

C. Decision function

The Decision function, depicted in the right part of Fig. 2, stores the kernel FIFO stream from the previous function in a BRAM, computes the Decision based on (1), and stores the decision vector in another BRAM (Dec). For multi-class classification, the decision must be calculated for each pair of classes. The range and number of support vectors in each class are received from another input (range_SV). Based on the decision vector, the number of votes for each class is calculated and the majority vote determines the final prediction (Vote).

Decision computation consists of two main loops for one pair of classes. For the same reason described in Sec. III-B for the accumulation, we manually unrolled these loops with factor DF to increase the efficiency. DF is the third HLS parameter to control the level of hardware parallelism.

D. Fixed-point implementation

To design a fixed-point SVM accelerator, we converted the floating-point data received from the *Read* function to a fixed-point value in the *Kernel* function. We explored the accuracy loss in hardware by varying the bit widths of each variable. Once the optimum fixed-point precision for all variables are obtained, HLS synthesis tool estimates the performance. To obtain the optimum precision, we divided the variables into three main parts that are used in *Inputs*, *Kernel*, and *Decision* computations. For the inputs, we selected $< 10, 1 >$ ($< total, integer >$) for x, SV, $bias$ and $< 21, 5 >$ for α. For *Kernel* and *Decision* computations, we selected $< 21, 11 >$ and $< 28, 10 >$, respectively. For MNIST dataset, these values result in the minimum accuracy loss in hardware.

IV. Results

Using Sklearn and an SVM model with RBF kernel, we first trained a classifier for the MNIST dataset. After training, we obtained 16036 SVs with size 784 ($N_{SV} = 16036$, $N_f = 784$). The test vector, support vectors, and other coefficients are sent to the SVM accelerator for the classification. We used Vivado HLS 2018.2 and the low-cost Zynq SoC of the ZedBoard for the evaluation of the performance. In addition to the MNIST case, we measured the hardware performance for other data dimensions to compare with previous works.

To show the impact of HLS-controlled parameters we report processing time and resource usage in Tab. II. Our accelerator can be used in various FPGA platforms by tuning N to adapt to the maximum memory bandwidth and by tuning BF to optimize the processing time. For the MNIST dataset, DF has no effect as the latency of Decision is negligible.

The first four experiments in Tab. II are with floating-point computation. In this case we have $II_K = 5$ and a computation-bound performance. Starting from the first experiment (N=BF=4), we can see that by increasing BF first to 8 and then to 16, the latency is reduced. Notice the increase in the resource usage due to BF and the negligible accuracy loss in fixed-point design compared to the floating-point one.

TABLE II
MNIST DATASET: PERFORMANCE AND RESOURCE USAGE.

Experiment	1	2	3	fix1	fix2
N	4	4	4	8	8
BF	4	8	16	8	16
BRAM (%)	15	16	19	12	15
DSP48 (%)	20	23	28	7	11
FF (%)	8	10	13	11	16
LUT (%)	24	28	35	40	62
Latency (ms)	166.93	91.57	58.69	18.12	15.72
Accuracy (%)	98.56 (float)			98.55 (fixed)	

The last two experiments are with fixed-point computation. In this case II_K=1 and it is possible to have a memory-bound performance. Therefore, increasing N from 4 to 8, the maximum allowed by the Zedboard platform, can be helpful. When $N = BF$ (experiment fix1), although the pipeline is balanced, the Kernel iteration latency has an impact on the overall performance. Therefore, by increasing BF (experiment fix2), we obtain a further latency decrease.

TABLE III
COMPARISON OF DIFFERENT SVM KERNELS.

kernels	RBF	Linear	Poly	Sigmoid
Time (ms)	58.69	53.88	58.53	64.94
BRAM (%)	0	0	5	3
DSP48 (%)	23	9	24	31
FF (%)	6	3	6	9
LUT (%)	16	8	13	23

Tab. III compares the performance of different SVM kernels. Sigmoid kernel has the highest resource usage and time, which is related to its computational complexity. Tab. IV shows a comparison of different HLS-based SVM accelerators in Zynq FPGA. Our dataflow design improves by about $10\times$ the processing time with less BRAM usage and more LUTs. In Tab. V we compared the number of features, SVs, and processing time for our design with two other FPGA accelerators designed manually in RTL. Due to the high scalability, our design can support higher number of SVs and features, with higher frequency and $4.4\times$ speed-up by using more resources.

TABLE IV
COMPARISON OF THE PROPOSED ACCELERATOR WITH DIFFERENT SVS AND SAME NUMBER OF FEATURES ($N_f = 27$) IN THE SAME FPGA (MODEL1 AND MODEL2 USE DIFFERENT PRE-PROCESSING METHODS ON TRAINING DATA).

	[12]	[13] (model1)	[13] (model2)	[14] (model1)	[14] (model2)	Proposed (dataflow)
$\mathbf{N_{SV}}$	248	61	248	248	346	**346**
freq (MHz)	100	250	250	250	100	100
Time (μs)	83.66	11.46	39.3	33.5	**136**	**13.15**
Interval (μs)	83.67	11.46	39.3	33.5	136	**8.15**
BRAM (%)	11	34	34	12	**11**	**2**
DSP48 (%)	61	2	2	61	61	88
FF (%)	13	10	28	13	13	30
LUT (%)	94	14	33	24	24	90

TABLE V
PERFORMANCE COMPARISON WITH TWO MANUAL RTL DESIGNS.

	[7]	[11]	This work
$\mathbf{N_{SV}}$	100	60	100
$\mathbf{N_f}$	500	1024	1024
FPGA	Virtex 5	Cyclone II	Zynq 7000
freq (MHz)	50	30	100
Time (ms)	0.25	2	0.45
#BRAM	-	-	39
#DSP48	52	20	83
#FF	9646	-	19687
#LUT	38179	14064	25758

V. Conclusions

We presented a scalable dataflow hardware architecture in FPGA by using HLS to accelerate SVM inference. The hardware parallelism can be controlled by three HLS-based configurations to adapt to small and large scale problems. In addition, a fixed-point design is introduced to speed up the computation. Experiments on different data dimensions and support vectors show a minimum of $10\times$ latency improvement compared to similar HLS-based and $4.4\times$ improvement compared to RTL-based designs.

Acknowledgment

This work was supported by the EMERALD project funded by the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No. 764479.

REFERENCES

[1] J. Cervantes, F. Garcia-Lamont, L. Rodríguez-Mazahua, and A. Lopez, "A comprehensive survey on support vector machine classification: Applications, challenges and trends," *Neurocomputing*, vol. 408, pp.189–215, 2020.

[2] M. Salucci, A. Polo, and J. Vrba, "Multi-Step Learning-by-Examples strategy for real-time brain stroke microwave scattering data inversion," *Electronics*, vol. 10, no. 95, 2021.

[3] L. Guo and A. Abbosh, "Stroke localization and classification using microwave tomography with k-means clustering and support vector machine," *Bioelectromagn*., vol. 39, no. 4, pp. 312–324, 2018.

[4] S. Afifi, H. GholamHosseini, and R. Sinha, "Fpga implementations of svm classifiers: A review," *SN Computer Science*, vol. 1, pp. 1–17, 2020.

[5] C. Kyrkou and T. Theocharides, "A Parallel Hardware Architecture for Real-Time Object Detection with Support Vector Machines," in *IEEE Transactions on Computers*, vol. 61, no. 6, pp. 831–842, June 2012.

[6] C. Kyrkou, C. Bouganis, T. Theocharides and M. M. Polycarpou, "Embedded Hardware-Efficient Real-Time Classification With Cascade Support Vector Machines," in *IEEE Transactions on Neural Networks and Learning Systems*, vol. 27, no. 1, pp. 99–112, Jan. 2016.

[7] M. Qasaimeh, A. Sagahyroon and T. Shanableh, "FPGA-Based Parallel Hardware Architecture for Real-Time Image Classification," in *IEEE Transactions on Computational Imaging*, vol. 1, no. 1, pp. 56–70, 2015.

[8] S. Saurav, R. Saini and S. Singh, "FPGA Based Implementation of Linear SVM for Facial Expression Classification," 2018 International Conference on Advances in Computing, Communications and Informatics (ICACCI), pp. 766–773, 2018.

[9] Y. Jiang, K. Virupakshappa and E. Oruklu, "FPGA implementation of a support vector machine classifier for Ultrasonic flaw detection," 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), pp. 180–183, 2017.

[10] Liu Han, Zhao Yue and Xie Guo, "Image segmentation implementation based on FPGA and SVM," 2017 3rd International Conference on Control, Automation and Robotics (ICCAR), pp. 405–409, 2017.

[11] M. Ruiz-Llata, G. Guarnizo and M. Yébenes-Calvino, "FPGA implementation of a support vector machine for classification and regression," The 2010 International Joint Conference on Neural Networks (IJCNN), pp. 1–5, 2010.

[12] S. Afifi, H. GholamHosseini and R. Sinha, "A low-cost FPGA-based SVM classifier for melanoma detection," 2016 IEEE EMBS Conference on Biomedical Engineering and Sciences (IECBES), pp. 631–636, 2016.

[13] S. Afifi, H. GholamHosseini and R. Sinha, "SVM classifier on chip for melanoma detection," 2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), pp. 270–274, 2017.

[14] S. Afifi, H. GholamHosseini, and R. Sinha, "A system on chip for melanoma detection using FPGA-based SVM classifier," *Microprocessors and Microsystems*, vol. 65, pp. 57–68, 2019.

[15] A. Baez, H. Fabelo, et al., "High-Level Synthesis of Multi-class SVM Using Code Refactoring to Classify Brain Cancer from Hyperspectral Images," *Electronics*, vol. 8, no. 1494, 2019.

[16] R. Campos and J. M. P. Cardoso, "On Data Parallelism Code Restructuring for HLS Targeting FPGAs," 2021 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW), pp. 144–151, 2021.