International Journal of Innovative Technology and Exploring Engineering (IJITEE)
ISSN: 2278-3075 (Online), Volume-8 Issue-12, October 2019

Hardware Acceleration of SVM classifier using Zynq SoC FPGA

Vidhyapathi CM, Maheshwar Reddy M, Nikhil Reddy T, Alex Noel Joseph Raj, Kathirvelan J

Abstract: ***Support Vector Machines (SVM) is one of the most commonly used the state-of-the-art supervised machine learning algorithm for various classification problems. It provides high accuracy rate compared to other classification algorithms. However, When SVM is modelled only using Software, it is a time consuming algorithm due to its high computational complexity. This makes the algorithm to be not suitable for embedded real time applications. We propose a new hardware software co-design approach to achieve the real time performance by accelerating the computationally intensive classifier part of the algorithm as a custom hardware Intellectual Property (IP) core. In this paper, a novel Support Vector Machine (SVM) linear classifier is modelled as a custom hardware Intellectual Property (IP) core using High Level Synthesis (HLS). The developed IP core is optimized for latency and hardware resource utilization by applying various directives of HLS tool. The synthesis results of the IP core for Skin segmentation dataset is reported. The proposed hardware software co-design approach is implemented in real time on Zynq-7000 XC7Z020 System on Chip (SoC) field programmable gate arrays (FPGA). A detailed comparative results of proposed hardware software co-design approach and the complete software approach is reported in this work for Iris and Breast cancer dataset. A promising result of 18x speedup is achieved using SVM classifier hardware IP compared to is software counterpart.***

Keywords: SVM, Hardware-software co-simulation, HLS, IP, SoC, FPGA.

I. INTRODUCTION

Support Vector machine is used for classification applications such as image classification, object detection, speech recognition, medical diagnosis and others [1, 2, and 3] with good accuracy. The SVM implements the two phases of Machine Learning: Learning the data and classifying the test-data based on the learning data. During the training phase of the SVM classifier, it develops a model, which is used to classify future dataset by making use of the Support Vectors, which were calculated during the training phase. The Support Vectors are then used to predict the class of the given test data instances. The SVM classifier as compared to other classifiers has shown better accuracy rates for many applications [4-7].However, the computation complexity of the SVM is directly proportional to the number of SVs and the size of the data sets. Computation complexity becomes the major bottleneck considering the real time embedded system based applications such as object classification, face recognition, automotive and embedded vision. On an embedded system platform which imposes tight constraints such as limited resources and area and power consumption, software modelling of SVM classification becomes difficult with more number of SVs. Also, modelling a time-consuming algorithms such as SVM for large-scale problems only on software will increase time complexity when datasets are large. Hence there is a need to analyze the computationally intensive part of the SVM and letting it to run on a dedicated hardware will speed up the algorithm at the cost of increased hardware. This idea motivated the researchers to accelerate the SVM algorithm using graphics processing units (GPUs) [3] and FPGAs [8]-[10].However GPUs suffers with more power consumption [11] and is not a preferable choice for embedded applications. Presently, FPGAs based heterogeneous computing platforms such as Zynq-7020 and customized hardware accelerators are the preferred choices [12] that consume less power and can be developed as a small system. Also, FPGAs provide high level of parallelism in processing which cuts down the computing time of a time complex algorithm. FPGAs in many applications have shown performance gain over general-purpose processors [13-17].

In this paper we have proposed a novel hardware-software co-design approach in which the computationally intensive vector dot product multiplication part of SVM on a custom hardware and rest of them on the software. The proposed hardware software co-design is implemented in real time using the Zynq-7020 SoC FPGAs.

II. RELATED WORK

All Research works until date have aimed at completely developing the classifier using HDL on FPGA. Pipelining technique [17, 18, and 19] was used in previous designs which exploit the parallel processing of the FPGA which led to increase the throughput of the classification process. Complete Software based modelling of SVM in many embedded applications such as pedestrian [20], [21], vehicle view detection [22] and face detection [23], [24] has shown good accuracy rates at the cost of limited performance.

Revised Manuscript Received on October 30, 2019.
***** Correspondence Author
Vidhyapathi CM*, SENSE Dept., Vellore Institute of Technology, Vellore, India. Email: vidhyapathi.cm@vit.ac.in
Maheshwar Reddy, SENSE Dept., Vellore Institute of Technology, Vellore, India. Email:maheswar.mannur@gmail.com
Nikhil Reddy, SENSE Dept., Vellore Institute of Technology, Vellore, India. Email: snthota1997@gmail.com
Alex Noel Joseph Raj, Department of Electronics Engineering, Shantou University, China. Email: jalexnoel@stu.edu.cn
Kathirvelan J, SENSE Dept., Vellore Institute of Technology, Vellore, India. Email: j.kathirvelan@vit.ac.in

Hence many researchers was motivated in accelerating the SVM algorithm through general-purpose processors, digital signal processors (DSPs) to achieve better performance on these hardware platforms.

In the work [25], the critical part of the algorithms were modelled using hardware and the rest in software. In work [26], an attempt was made to accelerate the SVM using a low power low cost 8 bit PIC microcontroller considering the limited memory and hardware resources. Graphics processing units (GPUs) was used in the work [27] due to the parallel nature of execution to improve the performance of SVM. However, the fixed hardware structure, need of efficient programming and high power consumptions of GPUs are the major drawbacks while using them for embedded systems.

Considerable work has been carried out by researchers in recent years, in accelerating the training and classification of SVM using custom reconfigurable computing hardware mostly on FPGAs. An analog-digital mixed SVM processor was proposed in the work [28]. In [29], a modified SVM training algorithm and the corresponding architecture was proposed. Small scale implementation of SVM with reduced number of SVs were proposed in [30], [31] and [32].Also these developed algorithms were strictly application specific and that were cannot be extended for other problems. In [33] and [34], the hardware acceleration of vector operations of SVM was analysed and better performance was reported. Furthermore, these work shown the importance of hardware accelerations using arithmetic and logic units (ALUs), multipliers and vector processing elements to achieve real time performance. All the related work was carried out keeping a specific application in focus. The proposed hardware for acceleration of SVM was not generic and was strictly application specific. In our work, we would like to propose a generic hardware software co-design approach. At the same time to provide flexibility in optimising the proposed custom hardware part of the SVM based on the specific application dataset.

To the best of our knowledge and survey, we believe that there exists no technique that makes use of hardware-software co-simulation technique exploiting the architecture of Zynq-7020.In addition, previous works were implemented in hardware definition language (HDL) alone, which was coded in Verilog or VHDL. This increases the design time for an application. We have made use of Xilinx tools such as Vivado HLS, which synthesizes C or C++ code to Verilog code and drastically reduces the design time for the application.

III. OVERVIEW OF THE PROPOSED METHODOLOGY

A. SVM Background

Support Vector Machine (SVM) was first introduced by Cortes and Vapnik in 1995 [35], [36], and is based on the concept of a decision boundary that separates two different classes of data in order to discriminate classes with high accuracy. SVM is a supervised ML (Machine Learning) classifier tool. They provide good performance for regression and classification tasks. A separating hyperplane is constructed in the training phase by using an input training data set containing data samples. The hyperplane that best separates the samples belonging to the two classes is called a maximum-margin hyperplane that forms the decision boundary. The class samples that are on the boundary are called Support Vectors (SVs) as depicted in Fig.1. These SVs obtained from training phase are then used in the classification phase to classify new data.

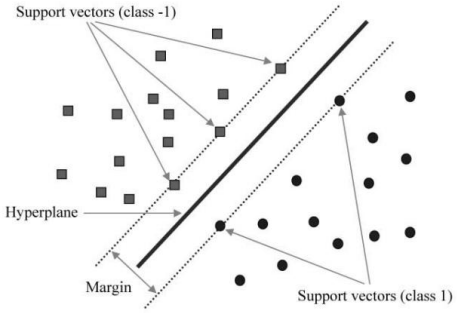

Fig. 1. Support Vector classes and hyperplane

A training dataset is initially used to model the decision function. This training dataset consists of input data vectors and its corresponding output class. The training process produces SVs which represent the entire dataset and can be used to classify any new data. The training or learning part is mostly an offline task while the classification is mostly online i.e. performed on real-time data. This classification phase is a computationally expensive task. Classification is linearly dependent on the test data size, its features and the number of Support Vectors. In such cases, there arises a need for acceleration of the classification process.

B. Hardware acceleration of SVM classifier using Zynq-7020 FPGA

Our aim is to make the time consuming part of the algorithm to run on the FPGA whereas the rest of the algorithm will be run on the Processing System of the Zynq SoC. To achieve this we make use of the Xilinx Vivado HLS tool, Vivado Design Suite and Xilinx SDK, Using the Vivado HLS we generate the Verilog or VHDL code by Synthesizing the C code or C++ code. To investigate the time consuming part the complete SVM is modelled in C code by giving the inputs as the model file. The predicted files are generated as the output file. Software profiling was done to understand the time consuming part of the SVM. Further we measured the time taken by the function which does the vector multiplication of the input data vectors which is time consuming part of the SVM algorithm, we measured the time to compare it later with the time taken on hardware alone.

The software modelled SVM has been verified by modifying the existing SVM-light library. We successfully classified different datasets such as Iris and breast cancel data sets and got high accuracy rates. To simplify the classification process and reduce the load of the computationally expensive process, we proposed a scalable, efficient hardware-software co-design approach, which loads the SVs, test data and the weights into an FPGA fabric to exploit its parallelization abilities. The burden on the CPU and the CPU processing time got reduced due to the parallel computation on the FPGA compared to the sequential execution of a traditional CPU.

Although SVM can be implemented in a variety of kernel tricks for linearly non-separable data, the basic core of an SVM classification phase is the vector multiplication of the decision function. Therefore, we chose to implement the linear kernel of SVM to achieve hardware acceleration.

We focus specifically on the acceleration of the classification phase by implementing the decision function, it makes use of factors found out during training phase namely alpha, bias and feature values , the number of support Vectors defined also determines the decision function which is used for classifying a given test data. The Eq. (1) is given by

$$F(x) = sign\left(\sum_{i=1}^{SV} a_i\, y_i (x_i . x) - b\right) \quad (1)$$

The above equation is implemented on FPGA by making use of the High level synthesis method which creates the IP for the top level function that implements the above equation. The decision function is divided into three equations.

$$AC = \sum_{i=1}^{SV} a_i y_i x_i \quad (2)$$

The above Eq. (2) multiplies the alpha value for each support vector with the respective number of features of each support vector and stores them in variable AC as shown above.

$$D = AC.x \quad (3)$$

By making use of AC found in Eq. (2), the variable D is the distance which is found by multiplying the AC value of each feature with the test data feature values as given in Eq. (3).

$$F(x) = sign(D - b) = \begin{cases} -1, & (D-b) < th \\ 1, & (D-b) \geq th \end{cases} \quad (4)$$

The calculated distance value is then subtracted from the bias value which was defined earlier during the training phase and the difference is used by the decision function to classify the test data to its respective class as given in Eq. (4).

The method we implement involves both software and hardware co-design, we are making use of the hybrid Zynq Architecture. The Zynq System [37] on Chip (SoC) consists of processing system (PS) with ARM cortex processor and a programmable logic (PL) on the hardware which is the FPGA.

C. Training SVM models on datasets

Before creating the SVM models on the Windows application, the required datasets need to be obtained. The datasets used in Machine Learning are mostly available either through Python libraries or in CSV formats. The above formats are difficult to be parsed in C due to the lacking of resources. Therefore, a Python script is written to download the datasets using import the file as CSV. The dataset is then normalized if required. The dataset is then separated into training and testing data. The training and testing data is written into files in the format they can be read by the SVM-light code. An example of how the SVM-light format of the dataset looks like is show in Fig. 2.

```
1 1:0.51 2:0.624 3:0.886
-1 1:0.78 2:0.773 3:0.639
-1 1:0.706 2:0.694 3:0.518
-1 1:0.004 2:0.012
-1 1:0.604 2:0.604 3:0.392
-1 1:0.6 2:0.612 3:0.431
1 1:0.404 2:0.549 3:0.816
-1 1:0.62 2:0.631 3:0.478
```

Fig. 2. SVM-light dataset format

The first column represents the class of the instance. Since the classification type is binary, the classes are represented as 1 and -1. The subsequent columns represent the value of the feature with its label. For example, in the first line, the first column, 1 means that it belongs to the first class. In this case of Skin Segmentation dataset, it means that the instance represents skin. The second column 1:0.51 means the first feature i.e. the colour Blue has a value of 0.51. The third column represents Green and the last column colour Blue.

The models are trained using the Windows SVM-light application for later classification on hardware. Cross-validation technique is used during the training phase to achieve a higher accuracy. Due to the limited runtime memory, the datasets were reduced in dimension by using scaling and normalization techniques.

The SVM-light was initially compiled and tested using the GCC compiler. It was later modified using the Visual Studio. Fig. 3. Shows the training process for the Fischer's Iris dataset using GCC compiler.

```
C:\Users\Sagar\Desktop\svm_lite>svm_learn iris\iris_train iris\model
Scanning examples...done
Reading examples into memory...OK. (90 examples read)
Setting default regularization parameter C=0.0197
Optimizing.............................done. (34 iterations)
Optimization finished (0 misclassified, maxdiff=0.00000).
Runtime in cpu-seconds: 0.00
Number of SV: 30 (including 28 at upper bound)
L1 loss: loss=4.27579
Norm of weight vector: |w|=0.69560
Norm of longest example vector: |x|=9.13674
Estimated VCdim of classifier: VCdim<=31.35710
Computing XiAlpha-estimates...done
Runtime for XiAlpha-estimates in cpu-seconds: 0.00
XiAlpha-estimate of the error: error<=31.11% (rho=1.00,depth=0)
XiAlpha-estimate of the recall: recall=>68.89% (rho=1.00,depth=0)
XiAlpha-estimate of the precision: precision=>68.89% (rho=1.00,depth=0)
Number of kernel evaluations: 1365
Writing model file...done
```

Fig. 3. Training Process for Iris Dataset using GCC compiler

After training, a model file is created which contains the information regarding the training dataset like the size of the data used for training. It also contains information like the number of Support Vectors and their values, number of features, the bias value and the type of kernel used for training. The first few lines of the model file are shown in Fig. 4.

```
SVM-light Version V6.02
0 # kernel type
3 # kernel parameter -d
1 # kernel parameter -g
1 # kernel parameter -s
1 # kernel parameter -r
empty# kernel parameter -u
4 # highest feature index
90 # number of training documents
31 # number of support vectors plus 1
-1.8553534 # threshold b, each following line is a SV (starting with alpha*y)
-0.019746901368946925825476057525552 1:5.8000002 2:2.5999999 3:4 4:1.2 #
0.019746901368946925825476057525552 1:4.6999998 2:3.2 3:1.6 4:0.2 #
-0.019746901368946925825476057525552 1:4.9000001 2:2.4000001 3:3.3 4:1 #
0.019746901368946925825476057525552 1:5.6999998 2:3.8 3:1.7 4:0.30000001 #
```

Fig. 4. Model File

D. Vivado HLS and SVM classifier Custom Hardware Intellectual Property (IP) core

Vivado High Level Synthesis (HLS) [38] uses C level source code to create an RTL implementation. The HLS tool extracts the control and dataflow from the source code. The design is implemented based on user-defined directives. This methodology allows for smaller, faster and more optimal design. High Level Synthesis is done mainly through scheduling and binding. Scheduling is mapping the sequential operations in the C code onto clock cycles. Binding is the mapping of the operations to the hardware cores. Fig.5. gives an outline of the steps involved in C code to RTL IP integration.

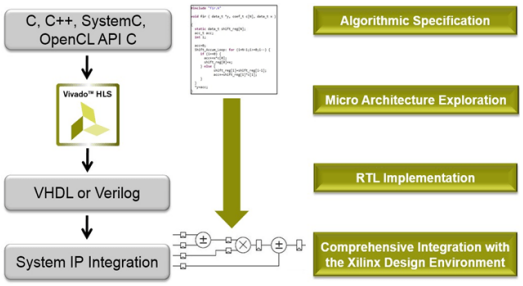

Fig. 5 HLS Design Flow for Algorithmic C to RTL IP integration [38]

The designer have complete control over the optimizations to be applied through directives like pipelining, unrolling loops, array partitioning, interfaces, etc. Vivado HLS supports higher level languages like C, C++ and SystemC provided that it is statically defined at compile time. If it cannot be defined until runtime, the code cannot be synthesized. In addition, there is support for the use of float and double in the code.

The HLS based custom created hardware IP block is shown in Fig. 6. The top level function contains the time consuming vector multiplication code of the SVM algorithm. The simulation and synthesis of the generated IP core is performed using the input and model data as test bench files. For achieving parallel processing, different directives are carefully examined and applied to achieve better performance with reduced hardware resources. Pipelining and unrolling are the specific directive from HLS which was used to optimise the "for loop" of the top-level function. After the successful C simulation and Synthesis, we generate RTL co-simulation and export the RTL into our custom IP block which creates a solution containing the equivalent Verilog code of the top level function.

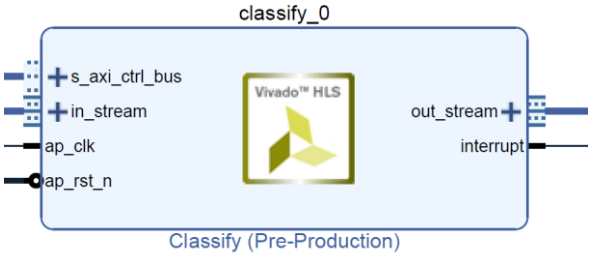

Fig. 6. Custom Hardware SVM classifier IP core

The C-based SVM classifier called SVM-light was used to implement it as an IP on Zynq-7020. Its implementation in C allowed us to use HLS methodology to speed up the IP creation process. The training part of the classification was done on Windows application using the basic parameters and the linear kernel. This model, which contains the support vectors and the information regarding the kernel and is later used by the IP to classify the test data.

E. Block design in Xilinx Vivado Design Suite

The Xilinx Vivado 2018.2 Design Suite was used to design and implement our hardware design on the Zed board i.e. Zynq-7020 Evaluation Board. The exported RTL design from Vivado HLS is added as
a repository in Vivado. Upon doing so, Vivado discovers the IP core. This IP core is added to the block design along with all the other blocks. The first one is the Zynq7 Processing System. Then the IP core, DMA and AXI Timer are added. All the necessary connections among PS, IP core, DMA, Timer and AXI interconnects are made. The S2MM and MM2S of the DMA and the out_stream and in_stream of the IP are interconnected to ensure the streaming of input and output data to and from between the IP core and PS.
The IP core in the Zynq-7020 PL is connected to the PS ARM through an ACP (Accelerator Coherency Port). This is preferable over the HP (High Performance) port as it reduces the burden of caching. This is because the ACP is a 64-bit slave interface on the SCU (Scoop Control Unit).This ensures cache coherent transmissions between the PL and PS, providing a direct low-latency path. All the transmissions at the ACP happen through the DMA core. Finally, the clock frequencies are set. Connection automation is run to connect the unconnected clocks and resets. Then the block is evaluated to make sure all the necessary ports are connected.
The block design connecting the Classify IP core to the PS and Timer can be seen in Fig. 7. The in_stream and out_stream between the IP core and the Zynq PS move through the AXI DMA. The IP core receives the in_stream as MM2S from DMA. The IP core sends the out_stream to the DMA as S2MM.
To measure the time taken to run the classifier IP in the PL, an AXI timer IP core was used while XTimer module was exploited to measure the clock cycles taken to run the same application on the PS part of the Zynq SoC. It was made sure that the models chosen for evaluation on Xilinx SDK tool were not so large. Although the application could be run for the large datasets on the PL part, the DDR3 memory's limitation meant that it could not be run on the ARM processor. Therefore, smaller datasets were chosen to evaluate the time measurements.

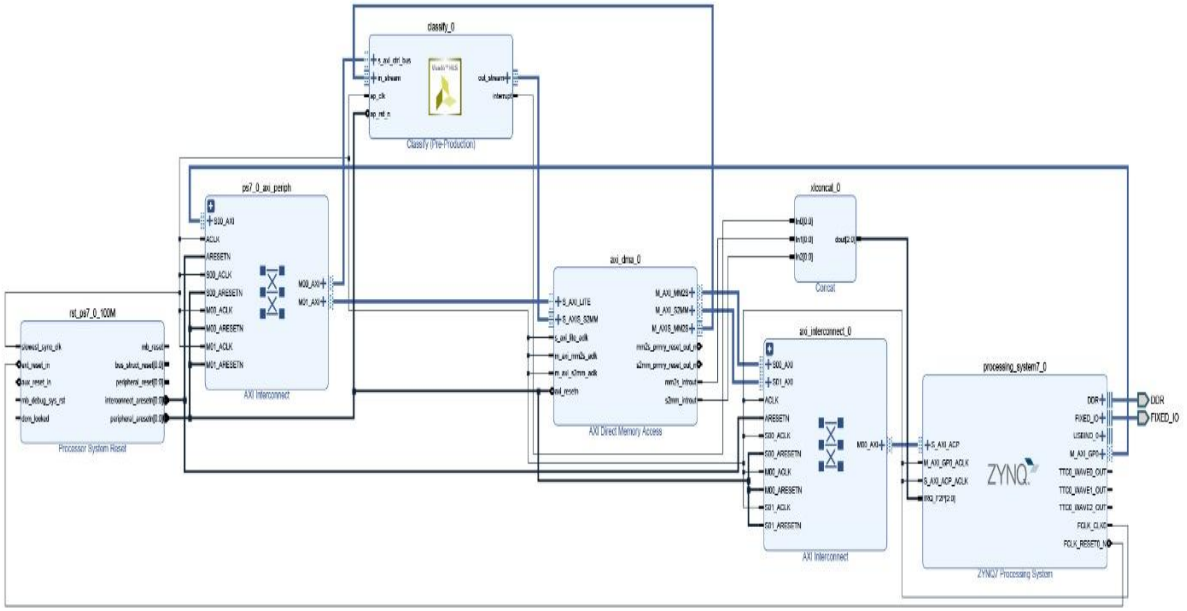

Fig.7. Hardware (Custom SVM classifier IP) and Software (ARM Processing System) co-block design

The final step is to generate the bit-stream from block design and use that in Xilinx SDK to dump the bit-stream onto the Zynq FPGA and verify the algorithm.

IV. RESULTS AND DISCUSSIONS

The SVM algorithm was first run on the modified version of the SVM-light C library using the GCC compiler from the windows command prompt for different datasets such as Iris dataset, handwritten digits, skin segmentation, breast Cancer and adult income datasets. Fig.8. shows the testing process for the Fischer's Iris dataset using GCC compiler. The accuracy of classification on the testing dataset is displayed.

```
C:\Users\Sagar\Desktop\svm_lite>svm_classify iris\iris_test iris\model iris\predictions
Reading model...OK. (30 support vectors read)
Classifying test examples..done
Runtime (without IO) in cpu-seconds: 0.00
Accuracy on test set: 100.00% (10 correct, 0 incorrect, 10 total)
Precision/recall on test set: 100.00%/100.00%
```

Fig.8. Testing Process for Iris Dataset using GCC compiler

After the testing process, the code creates a predictions file, which contains all the values before applying the sign function of the decision function. These values are used for assigning the class which the test instance belongs to. The prediction file for Iris dataset is shown in Fig. 9. The first five instances belong to the first class and the rest of them belong to the second class.

```
1.0096881
1.0511761
1.1015264
1.0651738
1.0744911
-1.0046381
-1.0499806
-1.17175
-0.29175611
-1.0114726
```

Fig. 9. Predictions File

The Table I depicts the classification accuracy achieved for each datasets.

A. Hardware resource utilization

The next phase was for implementation of the C algorithm in Vivado HLS, for designing the top level function and the test bench file and the data files. This generates the synthesis report and the equivalent Verilog code for the top-level function defined by the user. By making use of directives such as loop unrolling and pipelining we were able speedup the custom SVM classifier IP by reducing the latency through parallelism. Table II depicts the use of directives and the respective latency obtained for skin data set.

The first column of Table II represents the HLS directive applied on the IP. The remaining columns are the synthesis results generated by Vivado HLS. The first row of the table is after applying the basic interface directives (AXI4-Stream and AXI4-Lite). The subsequent rows are for all the other directives except the interfaces. After applying various directives to the design after interfaces, it can be seen that there has been a significant improvement in the latency and throughput. However, the improvement in latency and throughput has also resulted in utilization of additional hardware resources. The latency is seen to be significantly improved from 1,200,803 to 122,562 clock cycles, which is more than 9 times decrease. However, more number of resources were allocated which is an apparent trade off when improvement the latency. The large number of resources on the Zynq7 SoC is a reason why it is being used in this work to improve latency while being able to allocate the additional resources. However, we cannot simply go using a very large number of resources the reason being that the power consumption has to be kept at an optimum value. Therefore, a compromise has to be made among Latency, Hardware Resource Utilization and Power Consumption.

Table I. Classification Accuracy of different datasets

Dataset Used	Number of features	Training data size	Testing Data Size	Number of Support Vectors	Classification Accuracy (%)
Iris	4	90	10	30	100
MNIST (Digits)	154	69,000	1,000	19,530	89.4
Skin	3	22,051	24,506	5,211	92.68
Breast Cancer	30	512	57	137	89.47
Adult Income Prediction	123	30,956	1,605	11,066	82.74

Table II. Synthesis report for Skin Segmentation dataset

Directives applied	Latency(1 cycle = 10 ns)	Throughput	BRAM	DSP BLOCKS	Flip Flop	LUT
Basic Interfaces (AXI stream and Lite)	1,200,803	1,200,804	256	5	1,030	1,647
Array Partitioning (Cyclic factor 3)	784,197	784,198	192	11	1,373	2,174
Partial Pipelining of Outer Loops	147,067	147,068	192	10	1,713	2,314
Unrolling Inner Loops and Pipelining Outer Loops	122,562	122,563	192	10	1,701	2,348

Table III: Hardware resource utilization (Zynq-7010) of SVM model on different data sets

Hardware Component	Skin Segmentation data set	Iris data set	Breast Cancer dataset
FF	1,701 (1%)	893 (2%)	20,572 (19%)
BRAM	192 (68%)	0 (0%)	32 (11%)
LUT	2,348 (4%)	2,802 (5%)	16,756 (31%)
DSP48	10 (4%)	12 (5%)	12 (5%)

Hardware resource utilization of SVM model on Skin Segment dataset, Iris dataset and Breast Cancer dataset is reported in Table III. The hardware resource is mentioned in the first column and the subsequent columns show the number of the on-chip components being used and the percentage of the available resources. All these models are the results after applying the best optimization directives to achieve improvements in latency and resource utilization. As seen from the Tables 3, it is evident that the percentage of BRAMs used is high in the Skin Segmentation and Iris data set model. This is because arrays are implemented as BRAMs and since the support vectors and the test data instances are stored and passed as arrays, they require a high number of BRAMs for their storage. Here the latency is for the total number of clock cycles that were used to complete the process. There is an improvement in the latency when loop unrolling and pipelining are applied as directives. However, the hardware resources as we can see are more utilized when applying loop unrolling as it allocates more DSPs for the multiplications.

B. Processing Speed and Time

After Synthesis in Vivado HLS, using the Vivado Design suite the proposed hardware software co-design is validated and the final bit-stream is loaded into FGPA. Using Xilinx SDK, we developed an application to measure the processing speed and the time taken by the algorithm for each of the different datasets when run on software and compared those with hardware results. To measure the time taken to run the classifier IP in the PL, an AXI timer IP core was used while XTimer module was exploited to measure the clock cycles taken to run the same application on the PS part of the Zynq SoC. Initially a default clock frequency of 250 MHz was applied to both the PS and PL of the Zedboard. The following table depicts the clock cycles and the processing time taken by respective datasets. The results for this clock frequency configuration are provided in Table IV.

Table IV. Clock cycles and Processing Time comparison

Model	FPGA(1 cycle= 10ns)	ARM(1 cycle= 10ns)	Speedup Factor
Model 1	191 Cycles 0.76 µs	3455 Cycles 13.82 µs	**18.09**
Model 2	6759 Cycles 27.04 µs	131,057 Cycles 524.23 µs	**19.39**

Table V. Maximum frequency comparison

Data Model	FPGA	ARM	Speedup factor
Model 1	194 cycles	1,269 cycles	**6.54**
	0.77 µs	1.90 µs	**1.46**
Model 2	6,752 cycles	45,779 cycles	**6.78**
	27.08 µs	68.66 µs	**2.53**

Table VI. Speedup Factor comparison of different frequencies

Frequency	Clock cycles speedup	Processing Time Speedup
250 MHz	**19.39**	**19.39**
250 MHz and 666.67 MHz	**6.78**	**2.53**

Here the model 1 is trained for Iris Data and model 2 was trained for Breast cancer data. The Iris dataset classification application for Zynq SoC in the SDK tool took 191 clock cycles to run on the PL. This also constitutes the streaming of the data to the IP through the DMA. However, the same application on the ARM processor took 3455 clock cycles. Therefore the hardware IP achieved more than 18x the acceleration compared to the similar C coded function on the ARM CPU. In addition, results were obtained by operating the PS and PL at their maximum frequencies i.e. 666.67 MHz for the PS and 250 MHz for the PL. The results for this configuration are provided in Table V.

A comparison was made finally between the PS and PL using their maximum operating frequencies and the default used frequencies to conclude which is a better alternative for the processing time and speed, the Table VI concludes this.

It is clear from the Table VI that when both the processor and the FPGA operate at the same frequency a better speed up factor is obtained as compared to when operating at maximum frequency.

V. CONCLUSION

This work initially set out to implement the ML classifier, SVM on an FGPA. The reason was to develop an efficient and faster generic embedded classifier. Existing literature on this topic was studied and the gaps in the research were identified. Therefore, we focused on implementing an embedded classification system with high accuracy, while keeping in mind the constraints of embedded systems. Various design and development tools were exploited to develop such a system.

An accelerator for SVM IP was designed which implemented an online, efficient and scalable model. The proposed model is less complex and more hardware-friendly than a software implementation of the same algorithm. Five models were initially trained and implemented on the SVM-light application. The datasets for Fischer's Iris, MNIST handwritten digit recognition (modelled as even-odd classification), Breast Cancer prediction, and Skin Segmentation and Adult Income prediction were modelled in Windows. Three of the above-mentioned five models were also simulated and synthesized in HLS however there is a limitation for implementing larger datasets such as MNIST (Hand-written digits) on hardware FPGA as the resources for the large dataset were not sufficient and would require additional hardware. Regarding acceleration, a maximum factor of **20x** was achieved on Zynq FPGA compared to the similar system running on software i.e. the ARM PS. At 250 MHz, the processing time of **0.77 µs** was achieved on the FPGA, compared to the **1.90 µs** obtained on the ARM processor.

Classification accuracy was to be kept same while achieving hardware-friendly system. Most of the existing implementations in the literature suffered from an accuracy loss while trying to achieve performance improvements. However, we achieved consistent classification accuracies throughout our work from Windows application to HLS synthesis to Hardware implementation. Therefore, our work is a scalable, generic embedded SVM classifier with no loss in accuracy meeting the critical embedded system constraints.

REFERENCES

1. J. Nayak, B. Naik, and H. Behera, "A Comprehensive Survey on Support Vector Machine in Data Mining Tasks: Applications & Challenges," International Journal of Database Theory and Application, vol. 8, pp. 169-186, 2015.
2. C. J. C. Burges, "A tutorial on support vector machines for pattern recognition," Data Mining Knowl. Discovery, vol. 2, no. 2, pp. 121–167, 1998.
3. B. Catanzaro, N. Sundaram, and K. Keutzer, "Fast support vector machine training and classification on graphics processors," in Proc. 25th Int. Conf. Mach. Learn., 2009, pp. 104–111.
4. P. Sabouri, H. GholamHosseini, T. Larsson, and J. Collins, "A Cascade Classifier for Diagnosis of Melanoma in Clinical Images," in 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC), 2014, pp. 6748-6751.
5. G. M. Foody and A. Mathur, "A Relative Evaluation of Multiclass Image Classification by Support Vector Machines," IEEE Transactions on Geoscience and Remote Sensing, vol. 42, pp. 1335-1343, 2004.
6. R. Entezari-Maleki, A. Rezaei, and B. Minaei-Bidgoli, "Comparison of Classification Methods Based on the Type of Attributes and Sample Size," Journal of Convergence Information Technology, vol. 4, pp. 94-102, 2009.
7. J. KIM, B.-S. Kim, and S. Savarese, "Comparing Image Classification Methods: K-Nearest-Neighbor and Support-Vector-Machines", Ann Arbor, vol. 1001, pp. 48109-2122, 2012.
8. S. Cadambi et al., "A massively parallel FPGA-based coprocessor for support vector machines," in Proc. 17th IEEE Int. Symp. Field Program. Custom Comput. Mach. (FCCM), Apr. 2009, pp. 115–122.
9. O. Piña-Ramírez, R. Valdés-Cristerna, and O. Yáñez-Suárez, "An FPGA implementation of linear kernel support vector machines," in Proc. IEEE Int. Conf. Reconfigurable Comput. FPGA's, Sep. 2006, pp. 1–6.
10. M. Ruiz-Llata, G. Guarnizo, and M. Yébenes-Calvino, "FPGA implementation of a support vector machine for classification and regression," in Proc. Int. Joint Conf. Neural Netw., 2010, pp. 1–5.
11. J. Fowers, G. Brown, P. Cooke, and G. Stitt, "A performance and energy comparison of FPGAs, GPUs, and multicores for sliding-window applications," in Proc. ACM/SIGDA Int. Symp. Field Program. Gate Arrays (FPGA), 2012, pp. 47–56.
12. M. P. Véstias, "High-Performance Reconfigurable Computing Granularity," Encyclopedia of Information Science and Technology, pp. 3558-3567, 2015.
13. M. Wielgosz, E. Jamro, D. Zurek, and K. Wiatr, "FPGA Implementation of The Selected Parts of The Fast Image Segmentation," in Studies in Computational Intelligence vol. 390, ed, 2012, pp. 203-216.
14. T. Saegusa, T. Maruyama, and Y. Yamaguchi, "How Fast is an FPGA in Image Processing?," in International Conference on Field Programmable Logic and Applications, FPL 2008 2008, pp. 77-82.
15. H. M. Hussain, K. Benkrid, and H. Seker, "The Role of FPGAs as High Performance Computing Solution to Bioinformatics and Computational Biology Data," AIHLS2013, p. 102, 2013.
16. K. Nagarajan, B. Holland, A. D. George, K. C. Slatton, and H. Lam, "Accelerating Machine-Learning Algorithms on FPGAs using Pattern-Based Decomposition," Journal of Signal Processing Systems, vol. 62, pp. 43-63, 2011.
17. L. Woods, J. Teubner, and G. Alonso, "Real-Time Pattern Matching with FPGAs," in 2011 IEEE 27th International Conference on Data Engineering (ICDE), 2011, pp. 1292-1295.
18. A. Eklund, P. Dufort, D. Forsberg, and S. M. LaConte, "Medical Image Processing on The GPU–Past, Present and Future," Medical Image Analysis, vol. 17, pp. 1073-1094, 2013.
19. S. Asano, T. Maruyama, and Y. Yamaguchi, "Performance Comparison of FPGA, GPU and CPU in Image Processing," in International Conference on Field Programmable Logic and Applications, 2009. FPL 2009, 2009, pp. 126-131.
20. C. Papageorgiou and T. Poggio, "Trainable Pedestrian Detection System," Int'l J. Computer Vision, vol. 38, pp. 15-33, 2000.
21. M. Oren, C. Papageorgiou, P. Sinha, E. Osuna, and T. Poggio, "Pedestrian Detection Using Wavelet Templates," Proc. IEEE CS Conf. Computer Vision and Pattern Recognition, pp. 193-199, 1997.
22. S. Agarwal and D. Roth, "Learning a Sparse Representation for Object Detection," ECCV '02: Proc. Seventh European Conf. Computer Vision, pp. 113-130, 2002.
23. E. Osuna, R. Freund, and F. Girosi, "Training Support Vector Machines: An Application to Face Detection," Proc. IEEE Conf. Computer Vision and Pattern Recognition, pp. 130-136, 1997.
24. H. Sahbi, D. Geman, and N. Boujemaa, "Face Detection Using Coarse-to-Fine Support Vector Classifiers," Proc. Int'l Conf. Image Processing, pp. 925-928, 2002.
25. R. Pedersen and M. Schoeberl, "An Embedded Support Vector Machine," Proc. Fourth Workshop Intelligent Solutions in Embedded Systems, pp. 1-11, 2006.
26. A. Boni, F. Pianegiani, and D. Petri, "Low-Power and Low-Cost Implementation of SVMs for Smart Sensors," IEEE Trans.Instrumentation and Measurement, vol. 56, no. 1, pp. 39-44, Feb.2007.
27. B. Catanzaro, N. Sundaram, and K. Keutzer, "Fast Support Vector Machine Training and Classification on Graphics Processors," Proc. 25th Int'l Conf. Machine Learning, pp. 104-111, 2008.
28. R. Genov and G. Cauwenberghs, "Kerneltron: Support Vector "Machine" in Silicon," IEEE Trans. Neural Networks, vol. 14, no. 5, pp. 1426-1434, Sept. 2003.
29. D. Anguita, A. Boni, and S. Ridella, "A Digital Architecture for Support Vector Machines: Theory, Algorithm, and FPGA Implementation," IEEE Trans. Neural Networks, vol. 14, no. 5, pp. 993-1009, Sept. 2003.
30. I. Biasi, A. Boni, and A. Zorat, "A Reconfigurable Parallel Architecture for SVM Classification," Proc. IEEE Int'l Joint Conf. Neural Networks, vol. 5, pp. 2867-2872, 2005.
31. O. Pina-Ramirez, R. Valdes-Cristerna, and O. Yanez-Suarez, "An FPGA Implementation of Linear Kernel Support Vector Machines," Proc. IEEE Int'l Conf. Reconfigurable Computing and FPGA's, pp. 1-6, 2006.
32. R.A. Reyna, D. Esteve, D. Houzet, and M.-F. Albenge, "Implementation of the SVM Neural Network Generalization Function for Image Processing," Proc. IEEE Fifth Int'l Workshop Computer Architectures for Machine Perception, pp. 147-151, 2000.
33. R. Roberto, H. Dominique, D. Daniela, C. Florent, and O. Salim, "Object Recognition System-on-Chip Using the Support Vector Machines," EURASIP J. Advances in Signal Processing, vol. 2005, pp. 993-1004, 1900.

34. H. Peter, G. Srihari, C. Durdanovic, V. Jakkula, M. Sankardadass, E. Cosatto, and S. Chakradhar, "A Massively Parallel Digital Learning Processor," Proc. 22nd Ann. Conf. Neural Information Processing Systems (NIPS), pp. 529-536, 2008.
35. C. Cortes and V. Vapnik, "Support-Vector Networks," Machine Learning, vol. 20, no. 3, pp. 273-297, 1995.
36. V. Vapnik, The Nature of Statistical Learning Theory. Springer- Verlag, 1995.
37. "ZedBoard Hardware User's Guide", [Online], Available:www.zedboard.org/sites/default/files/documentations/ZedBoard_HW_UG_v2_2.pdf
38. Vivado Design Suite User Guide High-level Synthesis (UG902 v2018.2) June 2018.

AUTHORS PROFILE

Vidhyapathi CM received the B.E. degree in Electronics and Communication Engineering from Anna University, India, in 2006 and M.E. degree in VLSI Design from Anna University, Chennai, India, in 2008. Since 2010, he has been a member of faculty in the Department of Embedded Technology, School of Electronics Engineering, Vellore Institute of Technology, and Vellore, India, where he is currently an Assistant professor (Senior). His research interests are hardware acceleration of algorithms using FPGA, Algorithm design and system level optimization of embedded systems and computer vision.

.

Mannur Maheshwar Reddy recently received his B.E. degree in Electronics and Communications Engineering from Vellore Institute of Technology (VIT), India, in 2019. His research interests include Computer Architecture, Algorithm Optimization and Computer Vision.

Sai Nikhil Reddy Thota a graduate from the year 2019 in Bachelors degree of Technology in the field of Electronics and Communication Engineering from Vellore institute of technology. His research interests are machine learning algorithm optimization using hardware FPGA.

Alex Noel Joseph Raj received the B.E. degree in Electrical Engineering from Madras University, India, in 2001, the M.E. degree in Applied Electronics from Anna University in 2005, and the Ph.D. degree in Engineering from the University of Warwick in 2009. From October 2009 to September 2011, he was with Valeport LTD Totnes, UK as Design Engineer. From March 2013 to March 2017 he was with the Department of Embedded technology, School of Electronics Engineering, VIT University, Vellore, India as a professor. Since March 2017, he is with Department of Electronic Engineering, College of Engineering, and Shantou University, China. His research interests include Machine learning, Signal and image processing, and FPGA implementations.

Kathirvelan J received the Ph.D. degree in Engineering from VIT University in 2016. He is currently working as Associate Professor in Department of Sensor and Biomedical Technology, School of Electronics Engineering in VIT University, Vellore, India. His research interests include collision avoidance, control engineering computing, field programmable gate arrays, handicapped aids, infrared detectors, speech processing, and virtual instrumentation.