ASTES

Advances in Science, Technology and Engineering Systems Journal
Vol. 5, No. 5, 237-244 (2020)
www.astesj.com
Special Issue on Multidisciplinary Sciences and Engineering

ASTES Journal
ISSN: 2415-6698

FPGA Acceleration of Tree-based Learning Algorithms

Haytham Azmi*

Microelectronics Department, Electronics Research Institute, Cairo, Egypt

ARTICLE INFO

Article history:
Received: 21 June, 2020
Accepted: 02 September, 2020
Online: 14 September, 2020

Keywords:
Acceleration
Machine learning
Decision tree
FPGA

ABSTRACT

Machine learning classifiers provide many promising solutions for data classification in different disciplines. However, data classification at run time is still a very challenging task for real-time applications. Acceleration of machine-learning hardware solutions is needed to meet the requirements of real-time applications. This paper proposes a new implementation of a machine learning classifier on Field Programmable Gate Arrays (FPGA). The proposed implementation utilizes the MicroBlaze soft-core processor on FPGA and uses the Advanced eXtensible Interface (AXI) bus to integrate the MicroBlaze with hardware peripherals. Experimental results shows that hardware-software co-design is a promising solution as it saves silicon area and provides a flexible configuration of decision tree algorithms at run time.

1 Introduction

The continuous growth of Internet of Things (IoT) applications that need real-time data analysis, such as video cameras in surveillance applications, live streaming videos on social media networks and sensor data from smart cities, is driving the demand for an efficient platform that can process this huge amount of data and discover hidden patterns.

When designers explore hardware options for big-data analysis, Graphics Processing Units (GPUs) and Field Programmable Gate Arrays (FPGAs) appear as two promising solutions for the implementation of machine learning applications. GPUs have been dominating the parallel computing market for a long time and their hardware solutions have been thoroughly tested in different industrial domains. However, FPGA solutions have been recently considered in many high performance Artificial Intelligence (AI) applications and showed an improved power consumption, compared to GPUs.

To choose the optimum implementation, designers need to answer several design questions and explore possible solutions at early design phases. Examples of these questions are: what is the maximum clock frequency that can be reached in the target hardware platform? How many General Purpose Input/Output (GPIO) pins that can be utilized by the target hardware platform? Does the target hardware platform support parallel processing and concurrency or not? what is the average power consumption of similar implementations on the target platform? etc. Designers must study these questions carefully at early deign phases. The choice of the hardware platform has a significant impact on the performance of the target applications.

FPGA-based implementations of machine learning classifiers provide an efficient solution for real-time applications that use deep learning algorithms for classification [1]. FPGA-based implementation helps system engineers design modular systems and execute several tasks concurrently, which helps in processing real-time data more efficiently. The implementation of machine learning classifiers on FPGA can be done in many different ways. This paper discusses these alternative implementation options and presents a case study of hardware implementation of a machine learning algorithm on FGPA.

This paper has two main contributions.

- First, we present a comparative analysis of different design approaches of machine learning classifiers in the literature. The paper discusses technical implementation challenges and proposes practical solutions that help designers better implement efficient classifiers.
- Second, we propose a hybrid implementation of a machine learning, tree-based classifier. The proposed implementation utilizes hardware-software co-design approach that integrates a soft-core microprocessor on FPGA. The soft-core is used along with other digital blocks to build an area-efficient machine learning classifier. As a proof of concept, we use a MicroBlaze Xilinx IP core on Virtex-UltraScale FPGA to run the main tree-based algorithm and report its implementation details.

The novelty of the proposed implementation is that it bridges the

*Corresponding Author: Haytham Azmi, Microelectronics Department, Electronics Research Institute, Cairo, Egypt, haitham@eri.sci.eg

Table 1: Some of the proposed work in the literature that discusses the hardware implementation of machine learning classifiers.

N	Dataset	Classification	Target Platform	Performance	Reference
1	IBM DataQuest Generator	Decision Tree (Gini calculation)	PowerPC	5.58 times	**[2]**
2	UCI repository	Adaboost algorithm	VHDL Generator	5 to 10 ns per decision	**[3]**
3	Spambase UCI repository	Decision Trees Majority voting	Xilinx Virtex 5 XC5VLX110T	N/A	**[4]**
4	EEG data in house	Soft Decision Tree	Xilinx Spartan 3	98.1%	**[5]**
5	From **[6]**	Decision Tree (image-based)	software tool	75 million samples per second	**[7]**
6	From **[8]**	C4.5 classifier	Xilinx Virtex-6 XC6LX670	97.92% accuracy	**[9]**

gap between hardware and software when dealing with machine learning classifier implementation. Our solution addresses several challenges related to creating a customized machine learning classifier on FPGA from a raw dataset file. We discuss these challenges in detail in this paper and explain how researchers can use our proposed implementation to build low-cost, high-performance classifier on FPGA.

The rest of this paper is organized as follows. Section 2 presents the related work in the literature. Section 3 discusses the hardware implementation of the decision tree algorithm. Section 4 presents the results of the experimental work. Section 5 draws our conclusions and suggests new directions for future work.

2 Related Work

Utilizing hardware solutions for machine learning classification has been a challenging task for many researchers and system engineers. Researchers targeted different platforms, such as microprocessor-based systems, FPGA-based designs and parallel computing techniques to design efficient hardware classifiers for real-time applications. Table 1 highlights some of the proposed work in the literature that discusses the hardware implementation of machine learning classifiers. In this section, we highlight examples of the related work in more details.

In [2], the authors used *Gini* score calculation to develop a decision tree model. Authors went through two phases to build a decision tree model. First, they chose the splitting attribute and a split index for the root. Then, they split the children records based on the the first phase's decision. The proposed implementation in [2] recursively repeated this process until a stopping criterion is met [2]. In practical applications, Conventional Neural Networks (CNN) are trained off-line using training datasets through the back-propagation process. Following that, the trained CNN models are used to recognize images using the feed-forward process. Since there is currently a trending approach in recognizing images in real-time applications, the execution time of the feed-forward process is the most significant factor when it comes to performance evaluation [10]. In [11], the authors presented a software tool that can automatically generate a tree classifier Verilog code from python scripts. Authors used SciKit-Learn machine learning library to build the trained model and calculate the threshold values for the tree nodes. Following that, the tool is used to generate a Verilog code for decision trees and random forests [11].

In [12], the authors presented an implementation of Q-learning with Artificial Neural Networks (ANN) on FPGA for real-time applications that have latency and power constraints. The proposed implementation reduced processing time by utilizing the parallel structure of FPGA logic units. In [12], the authors demonstrated the implementation of a single neuron Q-learning accelerator as well as Multilayer Perception (MLP) Q-learning accelerator on a Virtex 7 FPGA. The authors discussed the architecture of the implementation in their paper [12]. However, when they evaluated the performance, it was compared to a CPU-based implementation. The speedup in performance was completely expected because of the concurrency achieved by FPGA. The paper did not provide any information about the speedup in performance when compared to GPU implementation [12].

In [13], the authors proposed a method to optimize CNN by utilizing four techniques: 1) fixed-point quantization to minimize calculations, 2) approximation of activation function to reduce the complex mathematical operations, 3) pipelining and parallelization of loops and tasks to speedup execution time, and 4) memory reorganization to enhance fetch time. Authors targeted the LeNet-5 architecture, which is a neural network architecture for handwritten and machine-printed character recognition [14]. They implemented their proposed accelerator on a Xilinx FPGA and used Zynq-7000 platform. The paper reported an operating frequency at 166MHz, which is not a useful measure for high performance applications. However, the paper reported that the proposed implementation can reduce the consumed energy by 93.7% compared to a GPU implementation [13]. The authors in [15] designed a decision tree classifier to recognize letters and digits. Xilinx Zynq SoC is used by the authors as a target hardware platform and Vivado high level synthesis is used as a development tool with C/C++ synthesis. Authors verified the correctness of the generated HDL code using C/RTL co-simulation [15].

In [16], the authors proposed a pipelined design that is partitioned into two stages. The first stage of the proposed design determines the conditions of the decision tree for which the classification can be done. The second phase of the proposed design used a pipelined data path for parallel execution.

Other implementations have been also proposed in the literature. In [17], the authors presented an automatic modulation classifier to identify the signal modulation format for electronic applications. In [18], the authors presented a hybrid classification engine using CPU and FPGA shared memory.

Although there are several proposed solutions in the literature for the hardware implementation of machine learning classifiers, the FPGA design and implementation of such classifiers were not discussed in proper details. We try to address this gap in our paper to present a hardware-software co-design implementation of a machine learning classifier on FPGA.

3 Hardware Implementation of Decision Trees

Designers explored various ways to implement decision tree algorithms on hardware platforms. In this section, we first highlights the basic concepts of implementing a single node on FPGA and how the synthesis tool interprets the HDL model and convert it into logic gates. Then, we discuss the common methodologies used in hardware implementation of such algorithms.

Classifying instances based on a training dataset has been used extensively to mine hidden patterns within detests [19]. One of the most popular classification algorithms is the tree-based classifier, where the main goal is to construct a tree from a labeled training dataset. The decision tree consists of a root node and children nodes. Each node represents a test on a feature, whereas each branch represents the outcome of a test, and each leaf node represents a class label. Figure 1 shows an example of a decision tree.

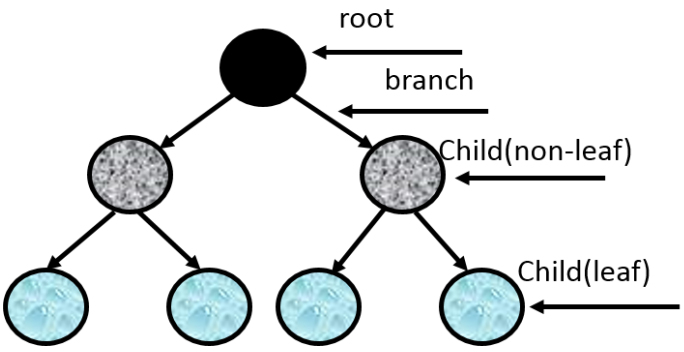


Figure 1: An example of a decision tree.

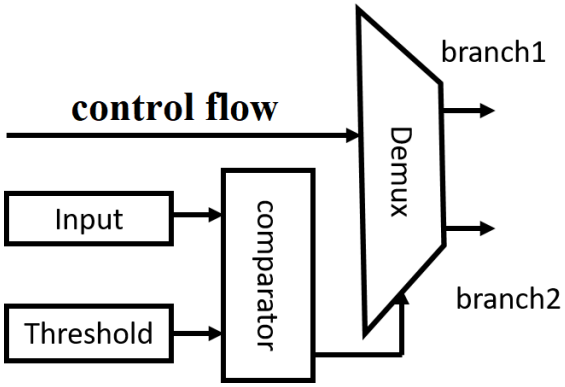
Decision trees can be constructed using different algorithms. The most critical design parameter is how to choose the feature at each node that best splits the dataset. The classification algorithms in the literature use different evaluation metrics to perform two design decisions.

1. Which feature should be used to split the dataset records at each node?
2. What is the the threshold value that should be used to split the dataset records at each node?

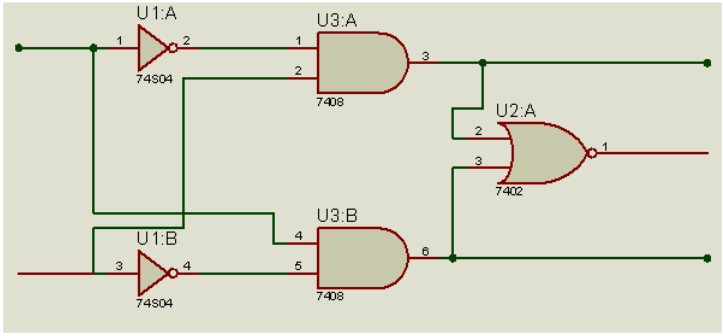
Examples of these evaluation metrics include using Gini index [20] and information gain, which is based on the concept of entropy [21]. Examples of decision tree algorithms include ID3, C4.5, CART, CHAID, and MARS [22]. ID3 is used in this paper as an example of a decision tree implementation. Other algorithms, such as C4.5 and CART, can be implemented using the same approach

A node in a decision tree algorithm represents a branching condition. If the input meets the condition, the flow of the tree goes through a certain path. Otherwise, a different path is selected. The tree can have a binary node or a multi-threshold node. Binary nodes branch out into two paths only, whereas multi-threshold nodes can branch out into more than two paths.

Figure 2 shows an example of the internal structure of a simple binary node. Figure 2a shows a block diagram of a branching circuit for a single node. The node first checks the input value and compares it to a given threshold value. The comparator then sends the result to a demultiplixer circuit that directs the flow of the control signals into one of two branches. A simple implementation of a one-bit comparator circuit is shown in Figure 2b. The comparator is usually designed to deal with n-bit values but Figure 2 is used here to simplify the concept.



(a) A block diagram of a branching circuit for a binary node



(b) Equivalent schematic of a branching circuit of a binary node

Figure 2: An example of a branching circuit of a binary node.

Since we are targeting Xilinx UltraScale architecture, we conducted an extensive study of the Configurable Logic Block (CLB) architecture of Xilinx UltraScale family. The CLB is the main building block that is used to implement all combinational and sequential logic of our proposed tree-base classifier. Each CLB unit consists of a group of logic elements along with an interconnect resource to connect all logic units. We followed the UltraScale Architecture Configurable Logic Block user guide in [23] to make sure that the synthesis tool will interpret the HDL code correctly and will generate an optimum logic design for the target application.

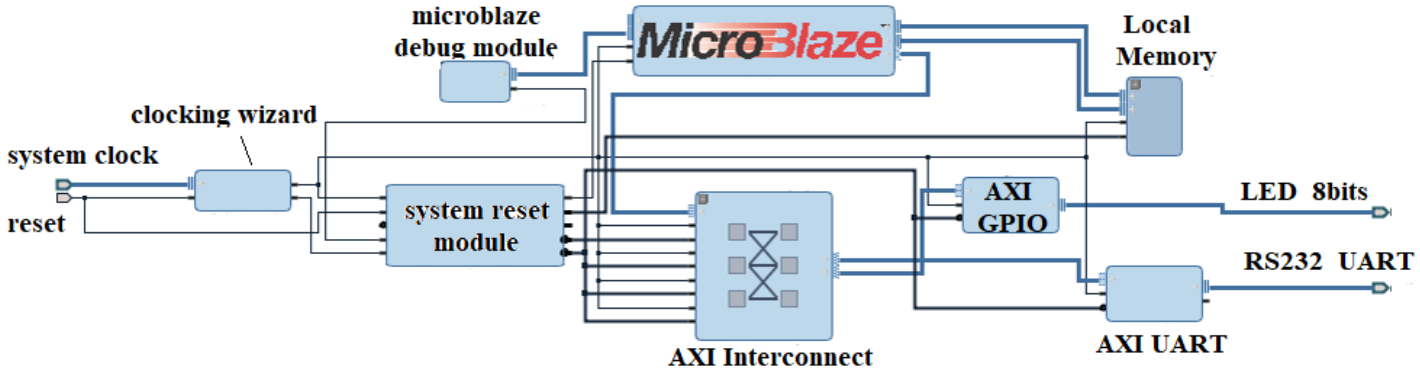


Figure 3: An example of a MicroBlaze interface circuit.

3.1 Implementation Challenges

Researchers utilized different techniques to implement machine learning algorithms on FPGA. Two common approaches are mostly cited in the literature. The first approach is writing the machine learning algorithm in a high level language, such as C, and then using a synthesis tool to convert from a high level language to a hardware description language. The second approach is targeting a structure-based design. Then, building all components in hardware and connecting these modules using port mapping. Each one of these two techniques has its advantages and disadvantages. Moreover, there are different implementation challenges associated with each approach. In this section, we discuss these challenges in details and explore different solutions to address them.

High Level synthesis (HLS) tools provide a mechanism to auto-generate HDL codes from high level languages, such as C, C++ and SystemC. HLS tools can save development time when implementing complex algorithms that can be easily represented in C language, for example. However, the majority of these tools do not provide complete support to generate recursion function in hardware description languages. Recursion comes with a difficult synthesis challenge by nature because the synthesis tool does not know when the exit condition will happen and hence can not predict the complexity of the hardware circuit that should be generated to implement the recursion function. For example, to generate a tree-based classifier, it is important that the synthesis tool knows the maximum depth of the tree. Otherwise, the tool can't support the generation process of the tree hardware logic at run time if the depth is unknown.

This limitation is blocking the usage of HLS tools in such scenarios. Researchers created various techniques to address this problem by developing recursive functions as an Embedded Domain Specific Language (EDSL) in C++ [24]. Authors in [24] addressed the recursion limitation by utilizing the C++ front-end of an HLS compiler [24]. However, this approach is not easy to integrate with HLS tools, such as Vivado, as the library needs to be re-customized every time the tool is used. This limitation makes it difficult to rely on this approach as a permanent solution, especially for real-time applications that require recursions to operate at run-time. Another proposed solution is having a maximum estimated depth of the tree and generating the digital logic for any tree-based classifier based on that maximum depth. However, this solution might fail if the target application requirements exceed the maximum depth. Moreover, if the requirements were within the maximum depth, the generated logic will have extra gates that are not used in the design, which causes an overhead area cost.

Another approach is utilizing structural implementation by dividing up the machine learning classifiers into separate blocks. These blocks are designed to deal with data-input, feature engineering, dataset training, and classification or prediction. For each block, designers write the hardware description code that implements the functionality of the intended block. The design is then connected together through port mapping as one complete design and used to process input data and solve machine learning problems.

This approach is proven to provide an optimized design for the target problem. However, the design process forces engineers to make very rigid design decisions to handle the target training set. For example, when generating a tree, designers can build a simple tree design and then re-instantiate this block through a pipeline structure. The challenge with this design method is that it causes a significant delay during the training phase because it executes the learning tasks sequentially. On the same context, designers can expand the tree to the required depth using hardware generators. This approach has a technology-based limitation as the maximum tree depth must be defined based on the target available silicon area.

3.2 Proposed Hardware/Software Co-design

Our proposed design aims at addressing the limitations of previous designs that are discussed earlier. We utilize the hardware-software co-design approach which gives designers the ability to execute several tasks in parallel while benefiting from the conventional implementation of complex recursive functions. This approach has been adopted previously by several researchers. The authors in [25] presented various implementations of decision trees using a sequence of universal nodes.

We explored soft and hard cores offered by Xilinx technology and we chose to use the the MicroBlaze IP core as our main processor [26]. MicroBlaze is built as a soft processor, RISC-based architecture, with a rich set of instructions that are optimized for embedded applications [27]. The MicroBlaze solution offers complete flexibility to select the needed peripheral, bus interfaces for memory and GPIO, and the ability to implement complex recursive functions that are needed in the machine learning training phase [27].

To complete the machine learning system design, we utilized other IPs to be able to train the network using different types of datasets. The soft-core is wrapped up by a Verilog code so that it can interact with the other components in the system. The soft-core is configurable in terms of number of ports, interrupt vectors, programmable timers, size of memory, and the maximum operating frequency.

Figure 3 shows a schematic diagram of the MicroBlaze interface circuit. The IP comes with default ports for interrupt, debug, clock , reset, DLMB, ILMB and AXI interfaces. Designers have several options when it comes to the configuration of the MicroBlaze interface. Designers can choose to integrate AXI or ACE bus and can also enable or disable the debug interface. To speed up the performance, designers can use instruction and data caches. The MicroBlaze is connected to its peripherals through an AXI interconnect.

Figure 4 shows that AXI interconnect can be configured in two different ways. First, the number of master/slave connections can be programmed to generate the required interfaces. Second, the interconnect optimization strategy can be set to minimize the area or maximize the performance, which gives the designer the ability to choose various strategies based on the application requirements.

Having a standard interface like AXI bus allows designers to easily integrate other hardware peripherals/IPs to the design easily. Since AXI is part of the Advanced Microcontroller Bus Architecture (AMBA) family, it will also easy to integrate any peripheral/IP that can be connected to AXI through bridges, such as APB, ACE, etc.

We conducted an extensive study of the architecture of the target FPGA to analyze the propagation delay through each LUT regardless of the function implemented, which is done for all classification functions.

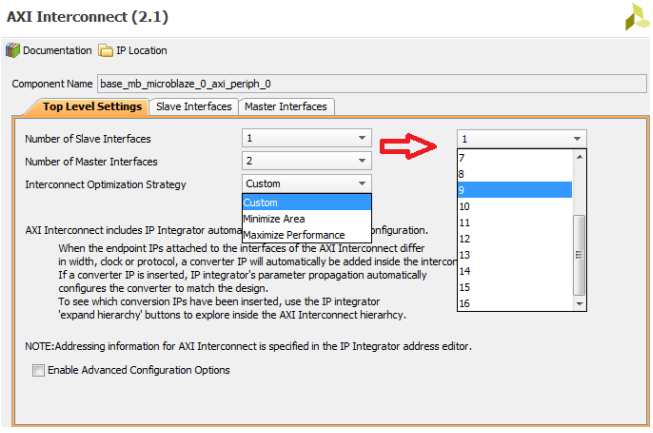


Figure 4: A programmable AXI interconnect control window from Xilinx.

To implement a hardware/software co-design on Virtex UltraScale, we use Vivado software tool. After completing the software programming in the Software Development Kit tool, we had to generate the .elf file and associate it with the MicroBlaze IP in Vivado to complete the integration process. The association process had several technical challenges. For example, the default block RAM size was one big limitation. We addressed this issue by allocating extra memory for the MicroBlaze software code.

The integration between hardware logic design and software code that runs on a microprocessor achieved several performance enhancement goals. First, the hardware logic of the entropy calculation and tree splitting is done using combinational logic gates that execute in parallel, which speeds up the processing of the tree generation during training and the classification during testing. The software code that runs on the MicroBlaze soft-core addressed the recursion task during the generation of the tree structure. Therefore, this integration was a major factor in improving the overall performance of the proposed design.

4 Experimental Work

The proposed system is designed to deal with different types of datasets that might need further processing. Figure 5 shows various data types that need to be considered before reading dataset for processing. Data can be qualitative or quantitative. Qualitative data includes

- nominal data, such as variables with no ranking sequence or inherent order like race and gender, etc.
- ordinal data, such as variable with an order like blood group, etc.
- binary data that has only two options, such as yes and no.

On the other hand, quantitative data includes

- discrete data, such as finite numbers of values that can not be classified meaningfully, etc.
- continuous data is information that can be measured on a scale and can be classified meaningfully, etc.

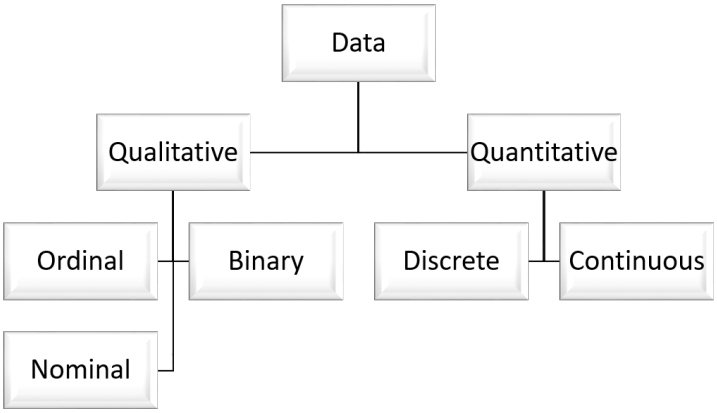


Figure 5: An explanation of the different data types that should be handled by the proposed design.

A pre-processing step must be executed first in order to convert the input data, into a standard format that is easy to process by the machine learning engine. One possible solution is to convert all various data types into binary format. This conversion is a very significant step to ensure that all input data types are converted to a machine-friendly format.

Figure 6 shows an example of the conversion process from nominal and numerical types of data to binary data type. The conversion

process is done as follows. First, each feature is examined to extract its list of all possible values. For example, the first column in Figure 6 (i.e. currency) can take three values: 1A, 5A and 6A. For each one of these values, we create a separate feature in the dataset in binary format. The new feature indicates whether each specific instance in the dataset has the currency set to that value or not. Since the currency feature has three values. Then, three new binary features are created to replace it. As shown in the example in Figure 6, the currency feature is replaced in the conversion by three binary features: currency(1), currency(5), and currency(6).

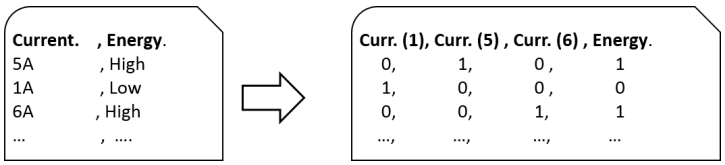


Figure 6: An example of converting nominal to binary format.

The number of features in Figure 6 for the output set will most likely increase based on how many nominal values are found in each feature. This conversion process helps researchers get an early estimation of the required memory blocks that should be allocated to host and process the training set.

4.1 Choosing the most significant feature

The most important decision in building a decision tree is choosing the root element that best distinguishes different patterns base on a certain threshold. The tree structure helps in building a model that classifies instances by navigating through various nodes in the tree until reaching one of the leaf nodes [28]. We use the information gain in this paper as our target evaluation metric so that the feature with the highest information gain is selected as the root element. The children nodes are then generated recursively. We continue to split the tree until we reach the decision nodes. The information gain calculations are based on the impurity level calculations in a set of samples. The formula used to calculate the entropy is given by [28]:

$$Entropy = -\sum_{i} P_i log_2 P_i \quad (1)$$

Where P_i is a probability of the class i relative to the total number of samples in the dataset.

The MicroBlaze processor is used to complete the calculations of the information gain as part of the software implementation of the target classifier. Feature selection plays a major role in improving the success rate of the decision tree classification. We conducted a comparative analysis to check which features are highly correlated and which features are causing overlaps that reduces the classification accuracy.

4.2 FPGA Implementation

FPGA implementation goes through various steps. First, we compile the code, then we simulate the design to make sure that the design functions correctly. Following that, we perform design synthesis to convert the design into equivalent logic gates. After synthesis, we perform post-synthesis simulation to ensure that the setup, hold, and propagation delays of the used gates don't affect the functionality of the design. Then, we perform the translation, mapping, place and route steps. Finally, we perform post-layout simulation and generate the bitstream file that is used to program the target FPGA.

Table 2 summarizes the results of the FPGA implementation of the proposed design. Due to the utilization of the MicroBlaze soft-core, we used limited resources from the target FPGA. Figure 7 presents a snapshot of the software tool that shows the placement and layout of the generated cells on the target FPGA. Figure 7 highlights the layout utilization percentage when implementing the proposed design in the target FPGA. The layout confirms the synthesis results that such a design doesn't require a lot of silicon area when implemented using the proposed SW/HW co-design approach. This is due to the utilization of the MicroBlaze software core, which saves a lot of silicon area during the implementation process. Figure 8 shows the statistics of on-chip components used from slice logic to implement the interface circuit with the MicroBlaze soft-core. It is clear from the figure that look-up-tables and registers constitute the majority components used from the slice logic in the FPGA.

Table 2: Implementation results

N	Item	Measure
1	Total On-Chip Power (W)	0.232
2	Dynamic (W)	0.154
3	Device Static (W)	0.151
4	Effective TJA (C/W)	1.7
5	Max Ambient (C)	84.5
6	Junction Temperature (C)	25.5
7	Slice Logic	3542
8	Block RAM	9
9	MMCM	2
10	I/O	14

We studied several implementations in the literature to compare the implementation of our proposed solution to previous implementations in the literature. We found out that it is difficult to make a one-to-one comparison due to three reasons. First, the target FPGA in the published work in the literature is different from ours. Therefore, the hardware utilization and used resources will not use the same base-reference to report the results, which makes it difficult to compare one implementation to another.

Second, there is no standard benchmark for dataset/classifier pair that is used across different implementations in the literature. Therefore, we could not implement a reference dataset/classifier pair to make a fair comparison. Third, the results are reported in different formats in the published papers. Few authors used utilization ratios, whereas others created their own cost unit formulas. Hence, the performance evaluation criteria are not the same across different publications. Although these factors affect the correctness and acc-

Table 3: Comparison with other implementation in the literature.

N	Implementation	Used Hardware	Implementation Main Finding	Reference
1	Gini-based decision tree	Xilinx Virtex-II Pro-based	31% minimum utilization	[29]
2	Hyper-rectangle hardware implementation	Xilinx Virtex (unknown version)	minimum 18 custom cost unit	[30]
3	Various architectures	Xilinx Virtex 5	SMpL needs 56% less resources	[31]
4	Multiple classifier systems,	Xilinx Virtex 5	95.8% accuracy - custom performance	[32]
5	Random Forest	GPU GeForce GTX 280	GPU implementation evaluates the forest in about 1% of the time compared to CPU	[33]
6	Random Forest	NVIDIA Tesla C2050	speed increases at least 300x compared to CPU	[34]
7	Proposed HW-SW co-design	Xilinx UltraScale	Less than 1% utilization of resources	This paper

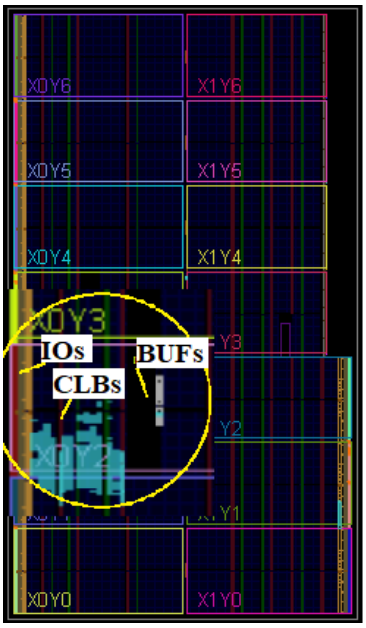


Figure 7: A snapshot of the FPGA placement and routing of the target design.

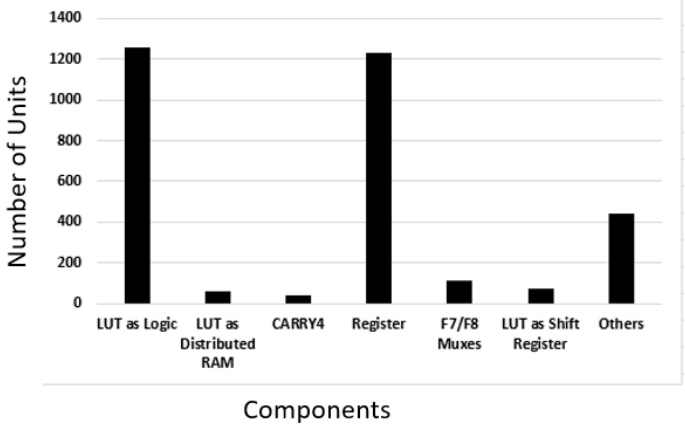


Figure 8: On-chip components used from slice logic.

curacy of the comparison, we reported the implementation results of various published work in Table 3. We presented these findings in Table 3 to show the different ways authors reported their implementation results of tree-based classifiers on FPGA. We also added GPU implementation results as examples of alternative hardware implementations. Experimental results show that the proposed design utilized less than 1% of the FPGA resources when implemented on Xilinx Virtex UltrScale FPGA.

5 Conclusion and Future Work

This paper presented a promising software-hardware co-design architecture for the implementation of machine learning algorithms. As a proof of concept, the paper discussed the implementation of a decision-tree algorithm based on ID3 implementation. The main functions of the ID3 algorithm ran on a MicroBlaze soft-core IP from Xilinx to utilize the processing units that are already available in the FPGA design flow. The integration of peripherals with the MicroBlaze is done through an AXI AMBA bus using a light weight version. Choosing AXI bus is an important design decision to make sure that the soft-core IP has a standard interface that simplifies the integration with other peripherals.

References

[1] A. Shawahna, S. M. Sait, A. El-Maleh, "FPGA-based Accelerators of Deep Learning Networks for Learning and Classification: A Review," CoRR, **abs/1901.00121**, 2019.

[2] R. Narayanan, D. Honbo, G. Memik, A. Choudhary, J. Zambreno, "An FPGA Implementation of Decision Tree Classification," in 2007 Design, Automation Test in Europe Conference Exhibition, 1–6, 2007, doi:10.1109/DATE.2007.364589.

[3] R. Narayanan, D. Honbo, G. Memik, A. Choudhary, J. Zambreno, "An FPGA Implementation of Decision Tree Classification," in 2007 Design, Automation Test in Europe Conference Exhibition, 1 – 6, 2007, doi:10.1109/DATE.2007.364589.

[4] M. Barbareschi, S. D. Prete, F. Gargiulo, A. Mazzeo, C. Sansone, "Decision Tree-Based Multiple Classifier Systems: An FPGA Perspective," International Workshop on Multiple Classifier Systems, 194–205, 2015.

[5] R.Harikumar, M.Balasubramani, "FPGA Synthesis Of Soft Decision Tree (SDT) For Classification Of Epilepsy Risk Level From Fuzzy Based Classifier Using EEG Signals," International Journal of Soft Computing and Engineering, **1**, 206–211, 2011.

[6] N. Dalal, B. Triggs, "Histograms of oriented gradients for human detection," in 2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05), volume 1, 886–893 vol. 1, 2005, doi: 10.1109/CVPR.2005.177.

[7] M. FULARZ, M. KRAFT, "Hardware implementation of a decision tree classifier for object recognition applications," Measurement Automation Monitoring, **61**, 379–381, 2015.

[8] T. Traces, "TCP Statistic and Analysis Tool," http://tstat.tlc.polito.it/, accessed: 2019-08-06.

[9] K. K. M. Da Tong, Lu Sun, V. Prasanna, "High Throughput and Programmable Online Traffic Classifier on FPGA," Proceedings of the ACM/SIGDA international symposium on Field programmable gate arrays, 255–264, 2013.

[10] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, J. Cong, "Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks," in Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays, FPGA '15, 161–170, ACM, New York, NY, USA, 2015, doi: 10.1145/2684746.2689060.

[11] T. Bhaaskar, K. Vishal, "Automatic generation of hardware Tree Classifiers," https://open.bu.edu/handle/2144/23688, 2017, accessed: 2020-09-07.

[12] P. R. Gankidi, J. Thangavelautham, "FPGA architecture for deep learning and its application to planetary robotics," in 2017 IEEE Aerospace Conference, 1–9, 2017, doi:10.1109/AERO.2017.7943929.

[13] Gan Feng, Zuyi Hu, Song Chen, Feng Wu, "Energy-efficient and high-throughput FPGA-based accelerator for Convolutional Neural Networks," in 2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT), 624–626, 2016, doi:10.1109/ICSICT.2016.7998996.

[14] Y. Lecun, L. Bottou, Y. Bengio, P. Haffner, "Gradient-based learning applied to document recognition," Proceedings of the IEEE, **86**(11), 2278–2324, 1998.

[15] R. Kulaga, M. Gorgon, "FPGA Implementation of Decision Trees and Tree Ensembles for Character Recognition in Vivado HLS," Image Processing and Communication, **19**, 71–82, 2015.

[16] F. Saqib, A. Dutta, J. Plusquellic, P. Ortiz, M. S. Pattichis, "Pipelined Decision Tree Classification Accelerator Implementation in FPGA," IEEE Transactions on Computers, **64**, 280–285, 2015.

[17] J. Grajal, O. Yeste-Ojeda, M. Sanchez, M. Garrido, M. Lopez-Vallejo, "Real-time FPGA Implementation of an Automatic Modulation Classifier for Electronic Warfare Applications," The 19th European Signal Processing Conference (EUSIPCO), 1514–1518, 2011.

[18] M. Owaida, H. Zhang, C. Zhang, G. Alonso, "Scalable Inference of Decision Tree Ensembles: Flexible Design for CPU-FPGA Platforms," 27th International Conference on Field Programmable Logic and Applications (FPL), 1–8, 2017.

[19] S. Surekha, "A Comparative Study of Rough Set Theoretic Decision Tree Induction Algorithms," in 2018 International Conference on Current Trends towards Converging Technologies (ICCTCT), 1–6, 2018, doi:10.1109/ICCTCT.2018.8550978.

[20] H. Liu, M. Zhou, X. S. Lu, C. Yao, "Weighted Gini index feature selection method for imbalanced data," in 2018 IEEE 15th International Conference on Networking, Sensing and Control (ICNSC), 1–6, 2018, doi: 10.1109/ICNSC.2018.8361371.

[21] B. Chen, L. Xing, B. Xu, H. Zhao, J. C. Príncipe, "Insights Into the Robustness of Minimum Error Entropy Estimation," IEEE Transactions on Neural Networks and Learning Systems, **29**(3), 731–737, 2018, doi:10.1109/TNNLS.2016.2636160.

[22] R. C. Barros, M. P. Basgalupp, A. C. P. L. F. de Carvalho, A. A. Freitas, "A Survey of Evolutionary Algorithms for Decision-Tree Induction," IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), **42**(3), 291–312, 2012, doi:10.1109/TSMCC.2011.2157494.

[23] "UltraScale Architecture Configurable Logic Block User Guide," *https://www.xilinx.com/support/documentation/user_guides/ug574-ultrascale-clb.pdf*, accessed: 2019-08-06.

[24] D. B. Thomas, "Synthesisable recursion for C++ HLS tools," in 2016 IEEE 27th International Conference on Application-specific Systems, Architectures and Processors (ASAP), 91–98, 2016, doi:10.1109/ASAP.2016.7760777.

[25] J. R. Struharik, "Implementing decision trees in hardware," in 2011 IEEE 9th International Symposium on Intelligent Systems and Informatics, 41–46, 2011, doi:10.1109/SISY.2011.6034358.

[26] H. Azmi, R. Sayed, "FPGA-based Implementation of a Tree-based Classifier using HW-SW Co-design," 2019 6th International Conference on Advanced Control Circuits and Systems (ACCS) & 2019 5th International Conference on New Paradigms in Electronics & information Technology (PEIT), 224–228, 2019.

[27] "Xilinx MicroBlaze Processor," *https://www.xilinx.com/products/intellectual-property/microblazecore.html*, accessed: 2020-08-26.

[28] Y. Zhong, "The analysis of cases based on decision tree," in 2016 7th IEEE International Conference on Software Engineering and Service Science (ICSESS), 142–147, 2016, doi:10.1109/ICSESS.2016.7883035.

[29] R. Narayanan, D. Honbo, G. Memik, A. Choudhary, J. Zambreno, "An FPGA Implementation of Decision Tree Classification," in 2007 Design, Automation Test in Europe Conference Exhibition, 1–6, 2007.

[30] J. Mitéran, J. Matas, J. Dubois, E. Bourennane, "Automatic FPGA based implementation of a classification tree," in SCS 2004 Signaux, Circuits et Systèmes Application to Informations Systems, IT 03, Workshop on Information Systems, 2004.

[31] J. R. Struharik, "Implementing decision trees in hardware," in 2011 IEEE 9th International Symposium on Intelligent Systems and Informatics, 41–46, 2011.

[32] M. Barbareschi, S. Del Prete, F. Gargiulo, A. Mazzeo, C. Sansone, "Decision Tree-Based Multiple Classifier Systems: An FPGA Perspective," in F. Schwenker, F. Roli, J. Kittler, editors, Multiple Classifier Systems, 194–205, Springer International Publishing, Cham, 2015.

[33] T. Sharp, "Implementing Decision Trees and Forests on a GPU," in D. Forsyth, P. Torr, A. Zisserman, editors, Computer Vision – ECCV 2008, 595–608, Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[34] D. Marron, A. Bifet, G. D. F. Morales, "Random Forests of Very Fast Decision Trees on GPU for Mining Evolving Big Data Streams," in Proceedings of the Twenty-First European Conference on Artificial Intelligence, ECAI'14, 615–620, IOS Press, NLD, 2014.