JETIR.ORG ISSN: 2349-5162 | ESTD Year : 2014 | Monthly Issue
JOURNAL OF EMERGING TECHNOLOGIES AND INNOVATIVE RESEARCH (JETIR)
An International Scholarly Open Access, Peer-reviewed, Refereed Journal

HIGH PERFORMANCE FPGA BASED TRAINING ACCELERATOR FOR TWO MEANS DECISION TREE

Mohammedsharukhpasha[1],Dr.B.NAGESHWARRAO[2] ,Dr.T.VAMSHI[3]

[1]M.Tech Student, Talla Padmavathi College of Engineering, Somidi, Kazipet, Telangana, 506003
[2]Assoc Professor, Talla Padmavathi College of Engineering Student, ,Somidi, Kazipet, Telangana, 506003
[3]Assoc Professor, Talla Padmavathi College of Engineering,Somidi, Kazipet, Telangana, 506003

[1]mohammedsharukhpash786@gmail.com,[2]nagesh.south@gmail.com, [3]vamshi22g@gmail.com

Abstract
Applications that make use of machine learning (ML) frequently make use of decision trees (DTs) because of the ease with which they can be interpreted and the speed with which they can be executed. Because DT training takes a significant amount of time, in this condensed version, we presented a hardware training accelerators as a means of accelerating the learning process. This field-programmable- gate -array (FPGA) with a maximum allowable frequency of 62 MHz is used to create the suggested training accelerator. The suggested architecture makes use of a combination of pipelined execution and parallel execution in order to reduce the amount of time needed for training and to use as little resources as possible. It has been determined that the C-based application software is at least 14 times slower than the proposed hardware design when it comes to a certain design. In addition, the suggested architecture merely requires a specific RESET signal in order to be readily retrained in order to process that this next set of data. Because of this form of training, the hardware can be used for a wide variety of different applications.

Key words: Decision tree (DT), field-programmable gate array (FPGA), machine learning (ML), training accelerator, two means DT (TMDT).

1. Introduction

Induction techniques based on decision trees (DT) segment the input space hierarchically in a top-down fashion and continue to do so until certain termination criteria are met. The data is routed through a decision boundary that is hosted at splitting nodes of the DT in order to achieve this hierarchical segmentation. At the leaf nodes, the tree completely loses its structure. These leaf nodes correlate to virtually uncontaminated

portions of the input space that have the vast majority of occurrences with almost the same label. The decision functions that are housed at splitting nodes are directly responsible for the increase in DT's training duration as well as the increase in the complexity of the model. This can range from straightforward axis-aligned rules and oblique splits to intricate networks (neural trees) and everything in between. The vast majority of real-world applications train DTs on some kind of software platform. The model parameters that are obtained through this kind of training are then included into the hardware chips that are used for activities involving embedded systems. However, retraining the characteristics on these chips can become challenging, making it tough to adapt them to a variety of applications.

In addition, numerous applications favour the locally available data over the embedded processors for reasons relating to mobility as well as privacy. In addition to this, an embedded device that is able to update its model has the benefit of being able to retrain itself based on newly acquired data. Graphical processing units, or GPUs, are frequently to use for machine learning (ML) programs that need intensive computations, which also results in intensive power consumption. In comparison, field-programmable gate arrays, more commonly known as FPGAs, are an accessible alternative that can speed up training in hardware. A number of different researches have been reported on different hardware architectures for the purpose of speeding up the DT classification. It has been proposed that the quick infrastructure for sifting sensor data by employing DT might be used. The DT & random forests classifiers were both built on the Von-Neumann CPU as well as the FPGA in this brief's suggested serial architecture, which then compared the two platforms' respective levels of efficiency. Even though it has a low power consumption, the FPGA implementation that was proposed in actually has a worse throughput. One of the FPGA implementations suggested by Tong et al. delivers balanced classifications in a smaller duration of time, and iteration generates an optimized tree in a smaller amount of space. Both of these implementations are described here. In order to reduce the amount of time spent classifying data, Saqib et al. presented a classification hardware that made use of pipelined architecture. There is a proposal for an Embedded system of an adjusting the amount for colour photos that uses k-means clustering. In this filter, data points are given to the cluster that is geographically closest to them..

2. Literature survey

“Progressively balanced multi-class neural trees”

Decision trees are a type of discriminative classifier that works by partitioning the input space in a hierarchical fashion in order to create regions that contain cases with the same class label. The majority of the previous research in this field has concentrated on C4.5 forests that have been taught orthogonally partitions. Neural trees, on the other hand, can learn oblique partitions off data and use a smaller number of specified points to host perceptions. On the other hand, these perceptions are prone to experiencing data imbalances. Because of this, we were inspired to suggest a gradually balanced neural tree, in which the training dataset is first made balanced before perception learning takes place. The modification of the decision surface with regard to entropy impurity-based goal functions constitutes the second contribution. In this formulation, it is also

possible for nodes to have a greater number of child nodes than two. On ten different standard datasets, the proposed approach is evaluated in comparison to three different baseline multi-class classification techniques.

"High-performance FPGA-based CNN accelerator with block-floating-point arithmetic"

CNNs are frequently employed in machine learning and voice processing and have shown to be quite effective. In the embedded system, however, the huge CNN model is constrained by computing and memory. In this paper, we use an optimised block-floating-point (BFP) arithmetic for efficient forecasting of deep neural networks. Off-chip memory stores feature images and model parameters in 16- and 8-bit forms, reducing storage and off-chip bandwidth needs by 50% and 75%, respectively, compared towards the 32-bit FP counterparts. An 8-bit BFP arithmetic with enhanced rounding and shifting-operation quantization techniques enhances power and equipment efficiency by a factor of 3. With a loss of accuracy of no more than 0.12%, a single CNN model can be used in our accelerators without the need for retraining. There are three parallel processing aspects, ping-pong off-chip DDR3 memory management, and an improved on-chip buffer group on the proposed reconfigurable accelerator. With an output of 768.3 GOP/s and 82.88 GOP/W, our accelerator greatly outperforms previous accelerators.

"Efficient hardware architectures for deep convolutional neural network"

The state-of-the-art deep learning approach is the convolutional neural network (CNN), which is used in a variety of applications. Real-time CNN solutions in embedded systems with minimal resources have recently become very popular. Using a field programmable array (FPGA) to create CNN models ensures programmability and shortens development time. However, the CNN acceleration is hampered by a lack of available bandwidth and even on memory storage. To speed up the performance of deep CNN models, we present efficient hardware architectures in this study. Parallel fast finite vector autoregressive algorithm (FFA) theoretical derivation is introduced. As a result of this work, CNN models can be constructed using fast convolutional units (FCUs). All transitional pixels are saved on-chip to reduce bandwidth requirements, and new data storing and reuse strategies are proposed. For our implementation of VGG16, we use a Zynq ZC706 & Virtex VC707 Xilinx FPGA. And used an analogous distance non-uniform quantization approach, we achieve top-5 accuracy of 86.25 percent. Xilinx ZC706 and VC707 are projected to perform at 316.23 and 1250.21 GOP/s, respectively, under 172-MHz and 170-MHz operating frequencies. In short, the proposed design greatly exceeds the existing works, particularly in terms of recycling and reuse, by more nearly two times.

3. Preliminaries

Because of the competing means DT (CMDT) approach, we developed the TMDT algorithm (from now on). As illustrated in the image below, the TMDT splitting node host two mean variables that determine the data transportation along the tree

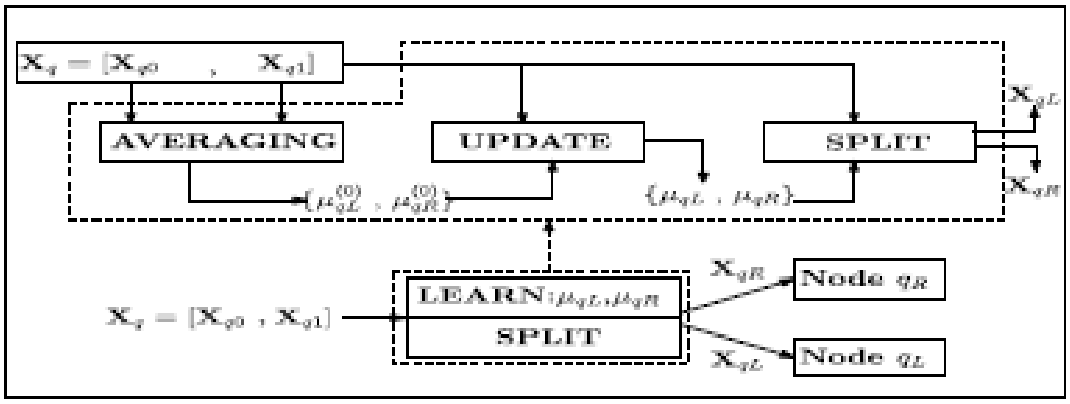


Fig 1. TMDT algorithm flow for a node *q*.

$\mathbf{X}_{q0} = \{\mathbf{x}^i : y(\mathbf{x}^i) = 0; \mathbf{x}^i \in \mathcal{R}^m; i = 0, \ldots, (n_{q0} - 1)\}$ and $\mathbf{X}_{q1} = \{\mathbf{x}^j : y(\mathbf{x}^j) = 1; \mathbf{x}^j \in \mathcal{R}^m; j = 0, \ldots, (n_{q1} - 1)\}$.

Consider the dataset Xq for node q to be [Xq0,Xq1] as a training set. y(x) is the reference group of x in this case. Learned from Xq are the mean matrices (split parameters). There are two stages to mastering the TMDT split node's settings. A starting point for constructing these models is to use the averages of the data points in Xq0 & Xq1, respectively. Discussed in this article of Xq are utilized to update overall mean vectors progressively in the second phase. There are two ways that are checked against each other to see how close they are to each other in terms of an instance's location. For simplicity, we will use the Distance measure of xk from $\boldsymbol{\mu}$ $(k-1)qL$ and $\boldsymbol{\mu}(k-1)qR$ as the units of measurement. This is how the (k - 1)th iteration's nearest mean vector is updated: xk.

$$\boldsymbol{\mu}_{qL}^{(k)} = \begin{cases} (1-\alpha)\boldsymbol{\mu}_{qL}^{(k-1)} + \alpha \mathbf{x}^k; & d_{qL}^{(k-1)} < d_{qR}^{(k-1)} \\ \boldsymbol{\mu}_{qL}^{(k-1)}; & \text{Otherwise} \end{cases} \quad (1)$$

$$\boldsymbol{\mu}_{qR}^{(k)} = \begin{cases} (1-\alpha)\boldsymbol{\mu}_{qR}^{(k-1)} + \alpha \mathbf{x}^k; & d_{qR}^{(k-1)} < d_{qL}^{(k-1)} \\ \boldsymbol{\mu}_{qR}^{(k-1)}; & \text{Otherwise.} \end{cases} \quad (2)$$

It is from these mean vectors produced at the end of phase two that child datasets XqL and the XqR datasets are created. When XqL and XqR are defined this way, they represent the instances that are closest to XQL and XQR, respectively. Additional split node properties can be learned from these data sets in order to better understand the parent-child relationships of the nodes on the right and left. Recursion is used to build a fully developed tree to a specified maximum depth dmax. A post pruning procedure is carried out on the mature TMDT. Pipelining of hardware platforms is made easier with this post-pruning technique

Leaf nodes are defined as TMDT nodes that meet one of its following conditions: first, based on purity criterion, if (max(nq0)/nq) surpasses a purity threshold; or second, using cardinality requirement, if (nq) falls below specified threshold nc. In the post pruning stage, all offspring nodes of a leaf node are wiped out. In addition, the majority class label of a leaf node is supplied as a parameter to this function. This is true for multiclass problems, regardless of their class names, as the instances will cluster into two groups. In the input dataset, leaf nodes will indeed be labeled according to their dominant class. Data input x is routed through to

the split cells by assessing its closeness to both of the two modes of data routing. This routing comes to an end for one of the intermediate node whose label is specified as the classifier of x.Splitting nodes in C4.5 learn an axis alignment rule of the type [l]_. The ideal split parameters (l,) are derived by grouping at every dimension while optimizing the impurities drop from parent to allowed students. For illustrate, with in case of Xq, a total of nq m nq lognq sorted operations and nq m impurity estimations will be necessary. However, the TMDT does a two-way clustering of the data set.

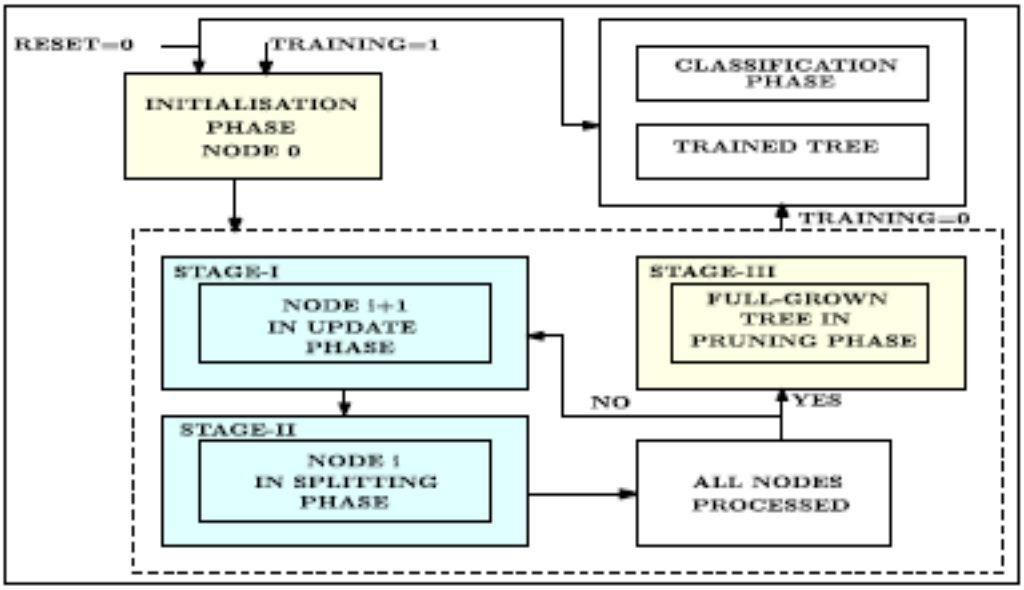


Fig 2. Training process flow of TMDT.

With only 4 nq comparison (two for each update and split) and nq updating processes needed, the TMDT nodes Q could complete all of its splitting with just four nq comparisons. Because of this, TMDT's latency is significantly less than that of C4.5. The sorting process necessitates a great deal of comparison. Sorting procedures take more time than updating operations because of the comparisons required. After trimming, each TMDT node's impurity is calculated only once. Next, the implementation of TMDT teaching is discussed.

4. Proposed system

The TMDT algorithm's training process flow is depicted in the previous chapter. There are three stages in the design process: standby, training, and categorization. FPGA power is turned on and active maximum signal RESET is declared 1, which sets hardware registers to zero in sleep mode. Training is activated once the TRAINING indication is set to 1 and the RESET is zero. To activate the categorization, the TRAINING signal must be zero after training is complete. RESET signals are ignored by the board while it performs categorization. Because when RESET is adjusted to 1, all prior data is wiped out and the equipment register contents are set to zero. By declaring TRAINING to 1 and RESET to 0, we begin training for new data. Once the training procedure again for new data is complete, the categorization begins. As a result, the RESET signal can be switched between 0 (training) and 1 (classification) to retrain this hardware on the new data (standby). Since it may be retrained at any time, the hardware created by translating this flow chart into architecture is particularly cost-effective. Because of its flexibility, FPGA is a better choice for implementing this type of

technology. Node 0 is loaded from on-chip storage to data register in the startup phase, as depicted in Fig. 2, and the initial mean is computed from the data. Initialization is completed in stage-I, and then stage-II is entered via pipelined registers by running node 0. The first mean computation of its leaf node, i.e. nodes 1 and 2, takes occur in parallel with splitting the data from node 0 when it is processed in stage-II. Node 1 enters stage-I, then stage-II, when the root node is performed. Stage-I supplies are released as node 1 joins stage II. As a result, node 2 is used in stage I whereas node 1 is put to use in stage II. The initial meanings of node 1's child nodes, namely nodes 3 & 4, are determined in parallel and placed in the node registry while node 1 is at stage II. Because node 3's initial mean is now in the node registers before node 2 enters stage-II, nodes 3 is passed to perform for the mean update. This means that the new mean of the (i+1)th node is already known, but the ith node is still in stage II, where I = 1,.., 14. Since the mean update and splitting phases are piped together, this will reduce resource consumption while enhancing throughput at the same time. Stage-III, the tree's last pruning, is performed once the tree has reached its full depth. Pruning conditions are compared to the amount of impurity & total data that enters a split node in this step. To signify that training is complete, the TRAINING signals is set to 0 and the hardware begins classification. Data is tagged with a label once it has been categorized using the decision rule obtained from the training tree.

4.1 Hardware Modules

The TMDT algorithm's hardware structure is displayed in the following figure, which was generated by mappings the flow diagram to an architecture. In the following, we'll go through some of the modules that make up this figure.

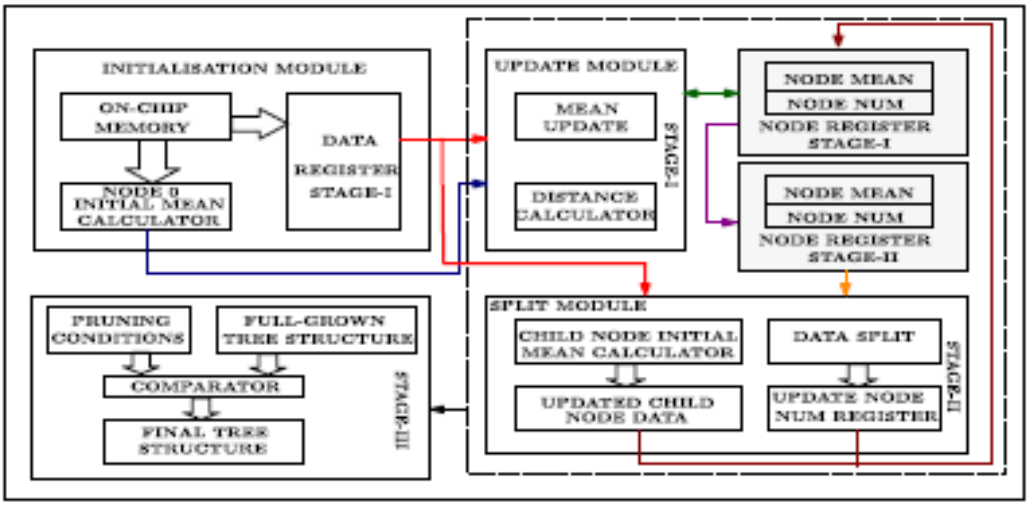


Fig. 3. Outline of hardware architecture for the TMDT algorithm.

Data are loaded into to the registers from the on-chip memories in the startup module to minimise the time lost in contacting the memory. Root node initial average is calculated in parallelism while the same datasets is transferred from on-chip random access memory into data register array (DRAM). A single call to the startup module completes the process. This is then used to update the root node's mean, which is calculated using the data registers array and the original mean. For all dimensions, the distance calculator module's architecture is parallel, therefore it's fast. This architecture utilizes m subtractors & m multipliers to parallelize the

computation of ([l]x[l])2, where l = 0,..., (m1). This means that it takes only tsub+tmul+tadd increments of time for the distance to be calculated. Thus, the extra time needed to calculate m dimensions sequentially is avoided by using this concurrent design. A distance calculator is used to calculate the distance between the data matrix's instances and the updated mean values, such that the left and right averages are updated based here on shortest distance. The nodes mean register is retrieved at the end of each iteration to store the revised mean calculated in stage-I of the pipeline. The node is transferred to the split module once all described in this article of the node have been handled in the update module.Split Module: As explained in Section II, the split module divides the node data into two parts depends on the difference from the left and right means. In the split subsystem, whenever an array of objects is assigned to a specific child node, the layer mainly of that data point is informed in the Base station Num register, and also the variable is decided to add to the preliminary mean of just that child node. Rather than having to calculate the initial average of each child node one at a time, this parallel processing saves time. The next node begins processing after all data has been separated and indeed the node registration array has been updated.

As soon as all nodes in stages I and II have been processed, stage III can begin. All of the full-grown tree's split nodes from the previous step are evaluated for the split situation in this stage. This is the final step. Leaf nodes are created for split branches that do not meet the splitting requirement, and their children are deleted from memory. It is then saved in memory when all nodes have been examined and the pruned decision tree has been established.

4.2 Hardware Architecture of Stage-I and Stage-II

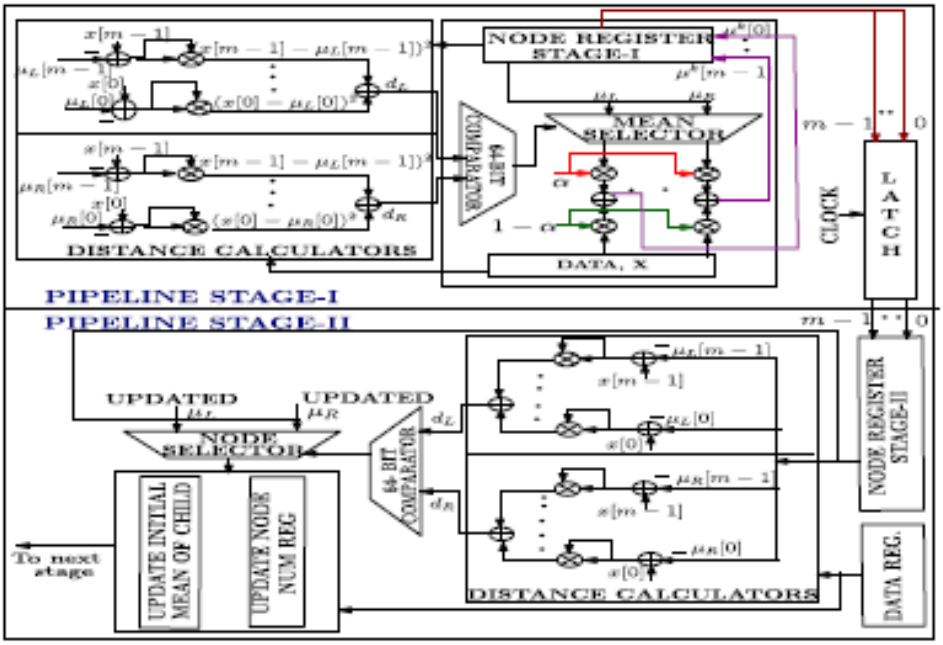


Fig. 4. Hardware architecture for pipelined stage-I and stage-II.

Above, you can see the stage-I and stage-II architecture in detail. In stage I, the distance in between data and the left and right means is calculated using two parallel distance calculator units. The mean selection compares the two distances and selects the mean with the smallest distance from data. The node register is then changed to reflect the newly determined mean. Load is defined to that node is processed in stage-I in this manner. This

is followed by a latching of the stage-I to stage-II transfer of the finalized mean and node numbers, after which stage-I proceeds to process the next node. To prevent data overwriting, the latch's clock period is set somewhat higher than stage-latency. II's Again, different distance calculators are employed in stage-II to compute the difference between xk and the updated node. The node identifier of the current node whose average is nearest to the data is used to identify the data. A register called Node Num keeps track of this number. As a result, each of the node's data instances is split in half and transmitted to one of its two child nodes. To compute the initial meant of a child node, all the data sent to that node is totalled up and entered into an initial mean register. Stage-III begins after stage-I and stage-II have been completed, and all nodes are processed.

4.3 Parallel Data Memory Access

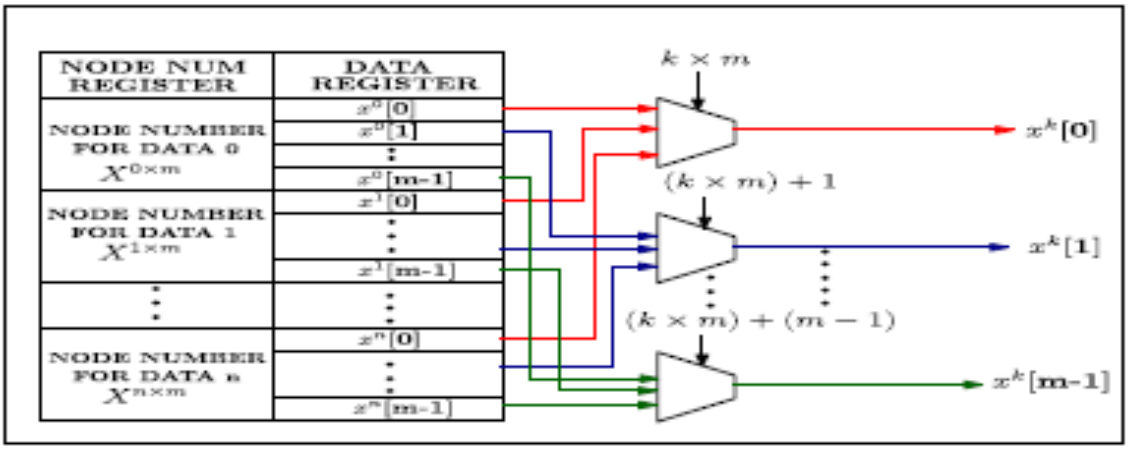


Fig. 5. Hardware architecture to access data memory registers in parallel.

For even more efficiency, as illustrated in the image above, data register accesses is parallelized. All of the data's node numbers are stored in a register called the Node Num record in identify which node they belong to. Node number (k+1) is examined if the node number of the kth data does not suit the current node number; alternatively, the kth data are accepted as the node number (k+1) does not match. Data from the current node is processed this way. m multiplexers are needed to simultaneously access all m dimensions of data. The choose line for identifying the 0th dimension of the kth data is the existing data location k m. It's a similar situation when you use the following formula: (k = m) + (m = 1). This saves (m 1) clock cycles by allowing access to mdimensional data in one place clock cycle. The data register array contains m memory addresses for each m-dimensional data set. The Node Num registers as well as the data register each hold 32 bits of information, making it possible to store n m data by utilising n m locations in the Node Num registration and n m locations in the data register. If a dataset is at location h, then the recorded information and its node number are at place h+m for such m-dimensional data, because data are stored sequentially.

5. Results

Fig. 6.Entity diagram for training accelerator two means decision tree is shown in below figure

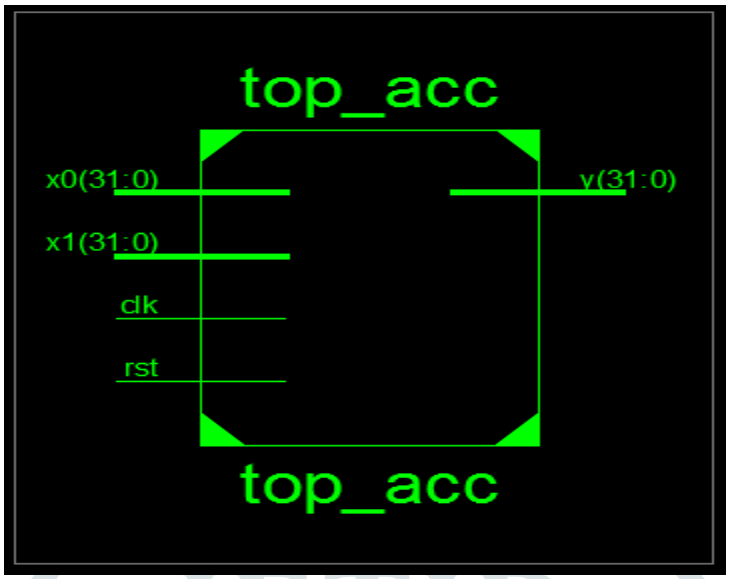


Fig. 7.RTL schematic for training accelerator two means decision tree is shown in below figure

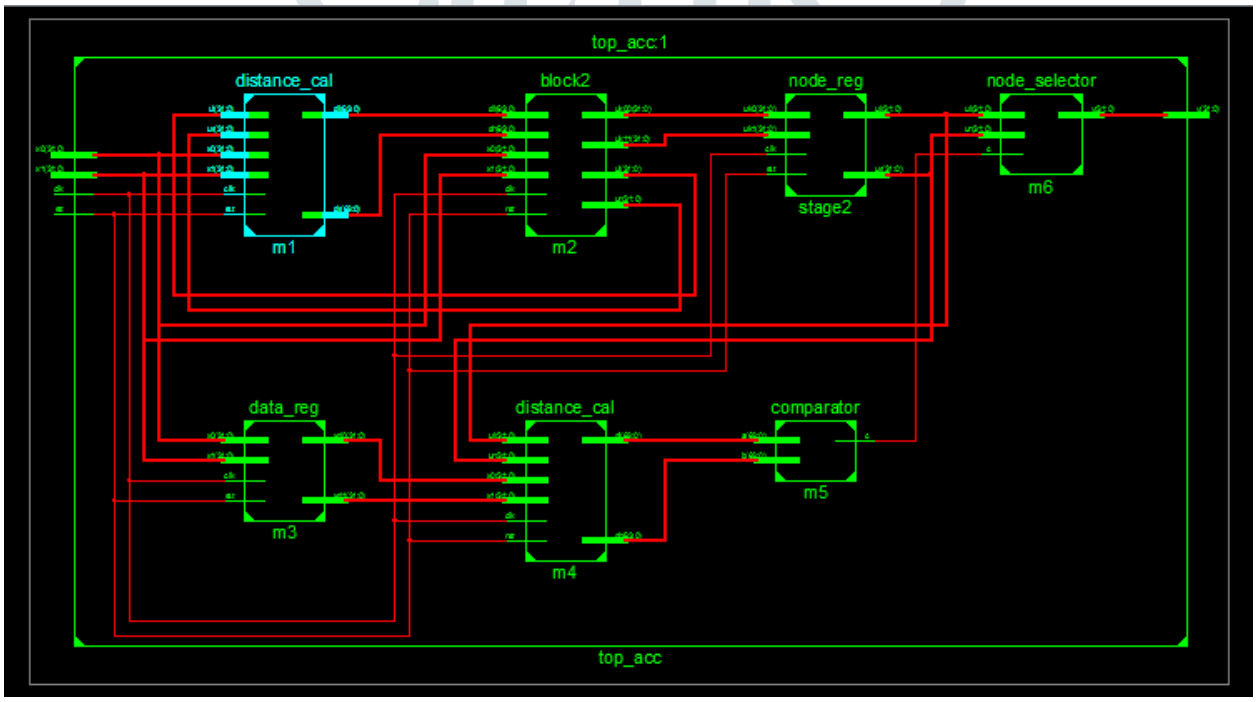
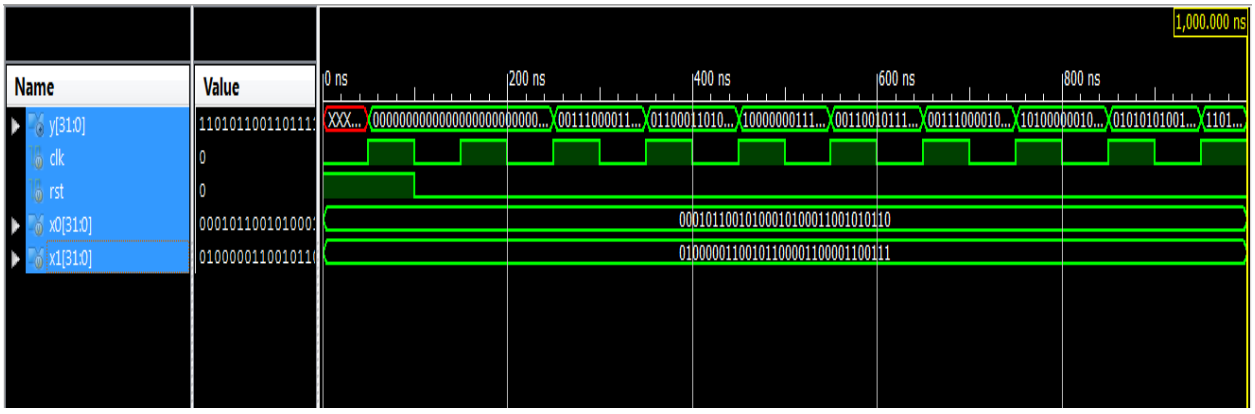


Fig. 8.Simulation results are obtained as follows



6. Conclusion

The TMDT algorithm has a training accelerator, which we describe in this brief. Fixed point & integer data were used to test the training hardware. An increase in training speed of at minimum 14 percent can be achieved using the training accelerator. A minimum of 224 103 hours of software training was necessary to finish the FPGA course, which was accomplished in 16.54 103. Multiclass classification will be implemented in batch mode on this hardware in the future to reduce the amount of memory used.Train for 32-bit data is supported by the hardware provided in this short. Both FPGA and CPU architectures were used to implement the TMDT algorithm. For purposes of software comparison, the algorithm was performed on a 3.2 GHz Intel Core i5 processor. Programming for the software was done in both Python and C. For the hardware implementation of the suggested accelerator, a Virtex Ultrascale+ XCVU9P-FSGD2104-3-E-ES1 FPGA boards was employed The design was developed using 16-nm technology and a performance grade of 3 to get the most out of it. Due to the wide main path of the size of 16 ns, the maximum operational frequency is 62 MHz. Most of this path's problems can be traced back to the frequent use of block RAM (BRAM). An on-chip 500 MB of RAM provides enough space to train 32-bit neural networks with about 125 million pieces of data. With such a massive on-chip memory, data may be accessed quickly and with little latency. On Vivado 2017, no high-level synthesizing tool was utilised to implement the design. An FPGA board's on-chip memory was preloaded with data from a PC before training began. By attaching the Electronic circuit direct current power source, this allowed for independent training. The architecture allows for up to two decimal places of 32-bit fixed-point implementation.

7. References

[1] S. Ruggieri, “Efficient C4.5 [classification algorithm],” *IEEE Trans. Knowl. Data Eng.*, vol. 14, no. 2, pp. 438–444, Apr. 2002.

[2] A. Godbole, S. Bhat, and P. Guha, “Progressively balanced multi-class neural trees,” in *Proc. 24th Nat. Conf. Commun. (NCC)*, Feb. 2018, pp. 1–6.

[3] X. Lian, Z. Liu, Z. Song, J. Dai, W. Zhou, and X. Ji, “High-performance FPGA-based CNN accelerator with block-floating-point arithmetic,” *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 27, no. 8, pp. 1874–1885, Aug. 2019.

[4] J. Wang, J. Lin, and Z. Wang, “Efficient hardware architectures for deep convolutional neural network,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 6, pp. 1941–1953, Jun. 2018.

[5] S. Buschjager and K. Morik, “Decision tree and random forest implementations for fast filtering of sensor data,” *IEEE Trans. Circuits Syst. I, Reg. Papers*, vol. 65, no. 1, pp. 209–222, Jan. 2018.

[6] D. Tong, Y. R. Qu, and V. K. Prasanna, “Accelerating decision tree based traffic classification on FPGA and multicore platforms,” *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 11, pp. 3046–3059, Nov. 2017.

[7] F. Saqib, A. Dutta, J. Plusquellic, P. Ortiz, and M. S. Pattichis, “Pipelined decision tree classification accelerator implementation in FPGA (DT-CAIF),” *IEEE Trans. Comput.*, vol. 64, no. 1, pp. 280–285, Jan. 2015.

[8] T. Saegusa and T. Maruyama, "An FPGA implementation of real-time K-means clustering for color images," *J. Real-Time Image Process.*, vol. 2, no. 4, pp. 309–318, Dec. 2007.

[9] G. Chrysos, P. Dagritzikos, I. Papaefstathiou, and A. Dollas, "HCCART: A parallel system implementation of data mining classification and regression tree (CART) algorithm on a multi-FPGA system," *ACM Trans. Archit. Code Optim.*, vol. 9, no. 4, pp. 1–25, Jan. 2013.

[10] F. Winterstein, S. Bayliss, and G. A. Constantinides, "FPGA-based K-means clustering using tree-based data structures," in *Proc. 23rd Int. Conf. Field Program. Log. Appl.*, Sep. 2013, pp. 1–6.

[11] S. Behnke and N. B. Karayiannis, "Competitive neural trees for pattern classification," *IEEE Trans. Neural Netw.*, vol. 9, no. 6, pp. 1352–1369, Nov. 1998.

[12] D. Dua and C. Graff. (2017). *UCI Machine Learning Repository*. [Online]. Available: http://archive.ics.uci.edu/ml