Implementing Decision Trees in Hardware

J.R. Struharik
University of Novi Sad, Faculty of Technical Sciences, Department of Electronics, Novi Sad, Serbia
rasti@uns.ac.rs

***Abstract*— In this paper several hardware implementations of decision trees (axis-parallel, oblique and non-linear) based on the concept of universal node and sequence of universal nodes are presented. Proposed hardware architectures are suitable for the implementation in both Field Programmable Gate Arrays (FPGA) and Application Specific Integrated Circuits (ASIC). Proposed architectures can be easily customized in order to fit a wide variety of application requirements, fulfilling their role as general purpose building blocks for System on Chip designs. Experimental results obtained on 23 datasets of standard University of California Irvine (UCI) Machine Learning Repository database suggest that the proposed architecture based on the sequence of universal nodes requires on average 56% less hardware resources compared with the previously proposed architectures, having the same throughput.**

I. INTRODUCTION

In the machine learning field, many different predictive models (classifiers), including the artificial neural networks (ANN) [1], decision trees (DT) [2] and recently introduced support vector machines (SVMs) [3] have been proposed.

Hardware realisation of ANNs received a significant attention in the scientific community, resulting in a number of proposed solutions [4-5], while SVMs hardware realisations have only started to appear more recently [6].

Decision trees (DTs) represent well-known predictive models, originally proposed by Breiman [9] more than 20 years ago. DTs are rooted tree structures, with leaves representing classifications and nodes representing tests of features that lead to those classifications. Typically DTs are implemented in software. Hardware implementation of DTs hasn't been investigated thoroughly so far. There are only two papers available in the literature, [7-8]. Paper [7] proposes hardware realisation of oblique DTs using the 2-dimensional systolic array architecture, while paper [8] presents the hardware realisation of axis-parallel DTs for a specific problem only (texture sea state classification).

In this paper, several digital architectures suitable for the hardware realisation of arbitrary DTs are proposed. These architectures can be easily customised to fit a wide variety of application requirements, making them good candidates for building blocks of System on Chip designs of embedded systems.

II. HARDWARE IMPLEMENTATION OF DECISION TREES

In DT learning, target function to be learnt is represented by a decision tree of finite depth. Every node of the tree specifies a test involving one or more attributes of the target function, and every branch descending from a node matches one of the possible outcomes of the test in the considered node. To classify an instance, a sequence of tests associated to the sequence of nodes is performed, starting with the root node and terminating with a leaf node. If numerical attributes are allowed only, the resulting DT will be binary. The majority of decision tree algorithms allow only numerical attributes.

Any test, generally speaking, can be written in the following way:

$$f\left(A_1, A_2, \ldots, A_n\right) = 0 \tag{1}$$

where f is a function of n attributes (A_1, A_2, A_3, ..., A_n). DTs can be classified according to the type of the function f. Thus, axis-parallel DTs [10], correspond to the function

$$f = A_i + a_i \tag{2}$$

oblique DTs (see e.g. [9], [11], and [13]) correspond to the function

$$f = \sum_{i=1}^{n} a_i \cdot A_i + a_{n+1} \tag{3}$$

and polynomial non-linear DTs (see e.g. [12-13]) correspond to the function

$$\begin{aligned} f_{2nd} &= \sum_{i=1}^{n} a_i A_i^2 + \sum_{i=1}^{n} a_i A_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} a_{i,j} A_i A_j + c \\ f_{3rd} &= \sum_{i=1}^{n} a_i A_i^3 + \sum_{i=1}^{n} a_i A_i^2 + \sum_{i=1}^{n} a_i A_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} a_{i,j} A_i A_j + \\ &+ \sum_{i=1}^{n} \sum_{\substack{j=1 \\ j \neq i}}^{n} a_{i,j} A_i^2 A_j + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \sum_{k=j+1}^{n} a_{i,j,k} A_i A_j A_k + c \end{aligned} \tag{4}$$

Oblique DTs normally lead to much smaller DTs compared to axis-parallel DTs. But finding the best oblique DT is an NP-complete problem, so oblique DT inducers normally use heuristics to find the best values for coefficients, a_i.

Since the form (4) can be embedded into the form (3) by introducing new set of artificial attributes which stand for the products $A_i^{m_i} \cdot A_j^{m_j} \cdot A_k^{m_k}$, provided that $m_i+m_j+m_k \leq 3$, non-linear DTs with polynomial hypersurfaces can be regarded as oblique DTs in higher dimension attribute space. Therefore the problem of

This work was partially supported by the Serbian MNTR grant No. TR32016.

finding the best polynomial non-linear DT is as hard as finding the best oblique DT.

According to the author's best knowledge, there are only few examples available in the open literature concerning the hardware realisation of DTs ([7-8]). A straightforward approach to the problem of hardware realisation of DT would be to implement every DT node as a separate module as it is proposed in [8]. This idea is illustrated in the Figure 1b, realising DT with three levels and five nodes (shown on the Figure 1a). A disadvantage of this approach is low throughput, due to the fact that new instance cannot be applied to the input before the completion of the propagation and appearance of the classification output for the previous instance.

Another example of the hardware realisation of DTs which is more advanced is proposed in [7]. This realisation, which is based on the equivalence between DTs and threshold networks, provides a rather fast throughput since the signals have to propagate through two levels only, irrelevant of the depth of the original DT. For example, a threshold network associated to the DT of Figure 1a is presented on Figure 1c.

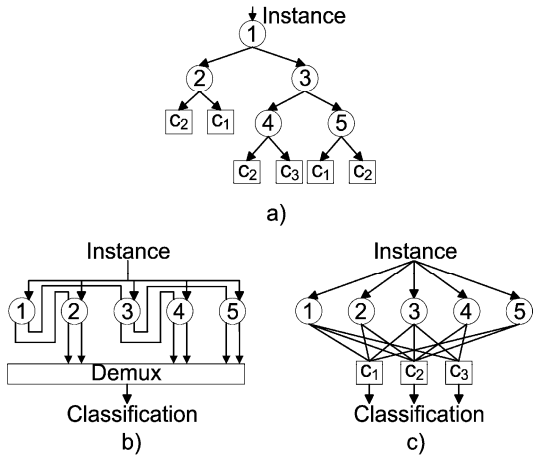

Figure 1. a) Sample DT of depth 3, b) Hardware implementation using architecture proposed in [8], c) Hardware implementation using architecture proposed in [7]

However, both architectures for hardware realisation of DTs mentioned above, require a considerable number of hardware resources (number of node modules to be realised is equal to the number of nodes in the DT). Finding an architecture that could ensure a substantial reduction in hardware complexity but preserve high throughput was the main idea behind this paper. An architecture which dramatically reduces hardware complexity (one universal node module only for whole DT), at the expense of lower throughput, has also been examined, which could also be interesting in some cases where classification speed is not critical. For the DT of Figure 1a, these two structures are presented on Figures 2a, b respectively.

During the classification of an instance using any DT, only a subset of nodes of DT which correspond to a path from the root node to a leaf of the DT will be visited. Therefore, to classify an instance, one node per level needs to be evaluated only, that is, number of node modules required for the realisation of a DT is actually equal to the depth of the tree. Each path has to go through each level, exactly once. Therefore it is enough to have only one universal node module per each level (this architecture is called Single Module per Level, SMpL).

In case when classification speed is not critical, a further reduction in the hardware complexity is possible: a single universal node can evaluate every tree node as shown on Figure 2b, independent from number of nodes and number of levels in the original DT (this architecture is called Universal Node, UN). UN architecture classifies instances in a sequential manner evaluating only nodes along the current path through DT levels from the root node to a leaf of the DT.

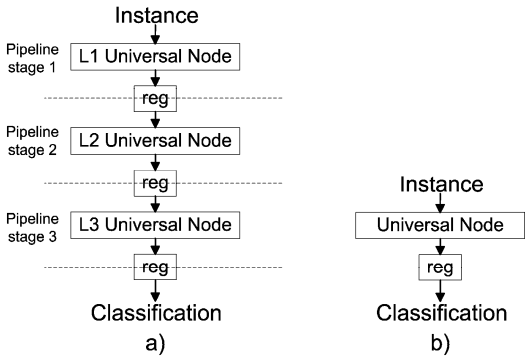

Figure 2. a) Basic structure of the Single Module per Level (SMpL) architecture for a DT of the depth 3, b) Basic structure of the Universal Node (UN) architecture

Following table summarizes the main characteristics of new and previously proposed architectures for the hardware realisation of DTs in terms of number of DT nodes (N_{dt}) and depth of DT (M).

TABLE I
HARDWARE RESOURCES, THROUGHPUT AND LATENCY FOR DIFFERENT ARCHITECTURES FOR HARDWARE REALISATION OF DTS

Architecture	Hardware Resources (# of node modules)	Throughput (# of instances per second)	Latency
proposed in [8]	N_{dt}	$1/M \cdot T_{node}$	$M \cdot T_{node}$
proposed in [7]	N_{dt}	$1/T_{node}$	$2 \cdot T_{node}$
SMpL	M	$1/T_{node}$	$M \cdot T_{node}$
UN	1	$1/num\_visited\_nodes \cdot T_{node}$	$num\_visited\_nodes \cdot T_{node}$

Table I shows that SMpL architecture has the same throughput as architecture proposed in [7], where T_{node} is the time needed to process instance in one DT node, but with substantial saving in hardware complexity measured as a ratio N_{dt}/M (the justification of using SMpL increases with the increase of this ratio). The largest saving, that equals to $N_{dt}/log_2(N_{dt})$, could be achieved in the case of complete binary DT. Notice that although SMpL and UN architectures reduce the number of node modules, that is, the number of adders and multipliers, the memory resources needed to store the hyperplane coefficient values remain the same as in the case architectures proposed in [7-8].

III. SINGLE MODULE PER LEVEL (SMPL) AND UNIVERSAL NODE (UN) ARCHITECTURES FOR OBLIQUE DECISION TREES

A. Details of Single Module per Level (SMpL) Architecture

SMpL architecture consists of *M* pipeline stages, *M* being the depth of the realized DT, as it is shown on the Figure 3. Each pipeline stage corresponds to a level of the DT.

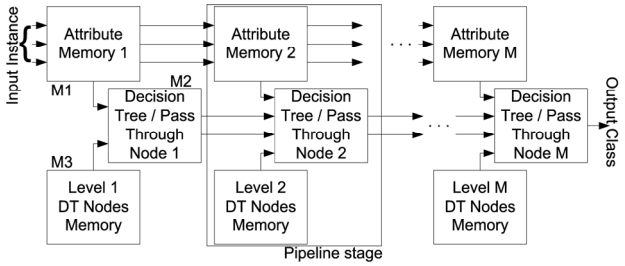

Figure 3. Details of the Single Module per Level architecture

To each DT level a pipeline stage is associated evaluating tests of the nodes positioned at that level. Each stage consists of three major modules: attribute memory, decision_tree/pass_through (DTPT) node and the memory storing the relevant information about the nodes from the same DT level. DTPT nodes and attribute memories form two pipeline chains of length *M*. Each attribute memory from the memory chain is connected to the corresponding DTPT node in the DTPT chain, as shown on Figure 3. Figure 4 presents a detailed architecture of pipeline stage.

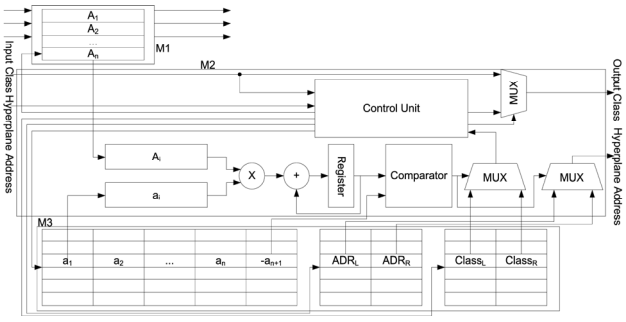

Figure 4. Detailed architecture of pipeline stage

Module M1 represents the attribute memory used to store the attribute values for the current instance. Its size is determined by the number of attributes, *n*, and the length of words, N_a, used to encode attributes.

Module M2 normally works in the decision tree mode and calculates both the position of the instance relative to the hyperplane associated to the selected node from the current level, and the address of the node from the next level to be visited. This mode is active when the value of "*Input Class"* port is zero, which implies that current instance hasn't been classified yet. Next node address will be transferred to the next pipeline stage through the *"Hyperplane Address"* output port. When a leaf is reached, module M2 calculates the class value of the current instance and transfers it to the next pipeline stage using the *"Output Class"* port. Otherwise, value of this port should be zero. In the case when the value of "*Input Class"* port is different from zero, module M2 transfers this value to the next pipeline stage through the *"Output Class"* port (module M2 works in the pass through mode).

Module M2 calculates instance position relative to the hyperplane sequentially using one multiplier and one adder (see equation (3).

Module M3 stores the information about the nodes from the associated level in the DT. It consists of three memory units. First memory unit having $(n+1)\cdot N_{dt}(l)$ locations, each N_c bit wide, where $N_{dt}(l)$ is the number of nodes at the level *l*, stores the hyperplane coefficients. This memory is organised into $N_{dt}(l)$ slices, each containing $(n+1)$ locations. Every slice stores the coefficients for one hyperplane.

Second memory unit stores addresses of the child nodes for all nodes of the associated level. Address value of every leaf node can be set to an arbitrary value.

Third memory unit stores class values of child nodes for all nodes belonging to the associated level. If a successor of the considered node is not a leaf its class value should be set to zero, to indicate that the current instance hasn't been classified yet.

A parallelisation in the evaluation of the test in each node could reduce the computational time from $n+1$ to one clock cycle only. This implies that the throughput of the SMpL architecture with parallelised evaluation would be one instance per clock cycle, which is clearly a significant improvement compared with the throughput of one instance per $n+1$ clock cycles in case of sequential evaluation. Clearly, this increase of throughput can be achieved only at the expense of increasing hardware complexity (parallel architecture requires *n* multipliers and $\lceil 2^{ld(n)+1} \rceil - 1$ adders instead of only one multiplier and one adder for the sequential architecture). In what follows, parallelized SMpL architecture will be called Single Module per Level Parallel, or in short SMpL-P.

B. Details of Universal Node (UN) Architecture

Universal Node architecture consists of one programmable node comprised of four modules, shown on Figure 5.

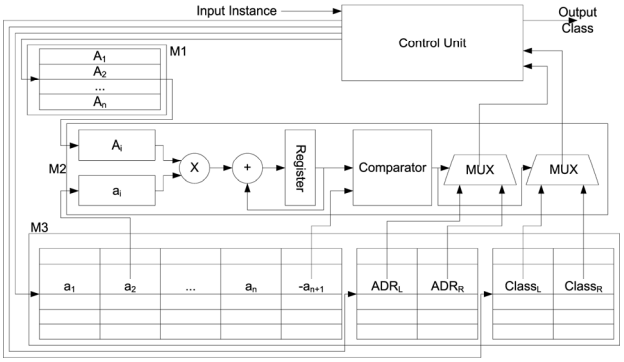

Figure 5. Details of Universal Node Architecture

Modules M1, M2 and M3 have the same function as respective modules in the SMpL architecture with two differences. First, module M2 in the case of universal

node architecture doesn't have pass through function. Second, address values of leaf nodes should be set to zero and class value for the non-leaf node can be set arbitrary in the module M3.

Fourth module is the control unit responsible for the correct operation of complete system. Control unit can operate in two different modes: one is used to load the attribute values of the next instance that needs to be classified and the other is used to classify the current instance.

A parallelisation in the evaluation of the test in the universal node could reduce computational time from $n+1$ to one clock cycle as in the case of SMpL architecture. Clearly, throughput of the parallelised UN architecture will be greater from the throughput of the sequential UN architecture by the factor of $n+1$. This modified UN architecture will be named Universal Node Parallel, or in short UN-P.

Architectures presented so far are all non-programmable, meaning that the structure of the implemented DT cannot be altered during the operation. If the DT structure needs to be changed, a new implementation must be generated. Although this might seem as a disadvantage, this is not so significant when FPGA technology is used, since it allows easy and quick reconfiguration of the device in the field. In case of ASIC technology programmable architectures are preferred.

IV. Required Resources

A. Configuring architectures for the hardware implementation of DTs

The well-know XOR classification problem will be used to illustrate how to configure proposed architectures for the hardware implementation of DTs. This problem is described by two numerical attributes and four 2-bit XOR instances that must be classified into one of two classes {1, 2}. Figure 6a shows the locations of four instances (marked with O's and X's) in the attribute space together with the classification regions obtained by one possible DT shown on Figure 6b. This DT would be created using some of the algorithms mentioned in Section II.

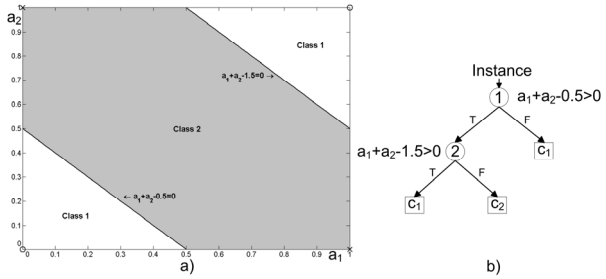

Figure 6. a) XOR classification problem with classification regions b) DT used for classification

As the illustration on how to configure SMpL architecture, Table II shows the content of memories needed for the realisation of DT of Figure 6b. Since depth of DT of Figure 6b is two, two pipeline stages are required. There is one node in the first level (root node), and one node in the second level of the DT.

TABLE II
CONTENT OF THE MEMORIES FOR ALL PIPELINE STAGES IN CASE OF REALISATION OF DT FROM FIGURE 6B USING SMPL ARCHITECTURE

Pipeline Stage	Coefficients Memory			Address Memory	Class Memory
Stage 1	*1*	*1*	*0.5*	*0&X*	*0&1*
Stage 2	*1*	*1*	*1.5*	*X&X*	*1&2*

In Table II symbol '&' represents the concatenation operator and X stands for the "don't care" value. Notice also that the free coefficient from the equation (3) is stored in the memory with the opposite sign.

For the realisation of the DT from Figure 6b using UN architecture, Table III shows the content of three memories.

TABLE III
CONTENT OF MEMORIES FROM M3 MODULE IN CASE OF REALISATION OF DT FROM FIGURE 6B USING UN ARCHITECTURE

Memory 1

Address	Value 1	Value 2	Value 3
0	1	1	0.5
3	1	1	1.5

Memory 2

Address	Value
0	1&0
1	0&0

Memory 3

Address	Value
0	X&1
1	1&2

B. Comparison of required resources and throughputs of SMpL, SMpL-P, UN and UN-P architectures

SMpL, SMpL-P, UN and UN-P architectures differ one from another in the hardware complexity and classification speed as shown in Table IV. The size of required memory, number of multipliers and adders and throughput for the considered architectures are expressed in terms of: the number of nodes of oblique DT (N_{dt}), depth of the tree (M), the number of problem attributes (n), the number of bits for representation of attributes values (N_a), coefficients values (N_c) and clock cycle period (T_{clk}).

Table IV shows that SMpL and SMpL-P architectures require only slightly more memory compared with the universal node architectures (UN and UN-P) in order to be able to store the attribute values in every pipeline stage as well.

In terms of the number of required adders and multipliers, SMpL-P architecture is the most demanding, while the UN architecture is clearly the least demanding, requiring only one adder and one multiplier.

When throughput is considered, SMpL-P architecture is the best one, classifying one instance per one clock cycle. SMpL and UN-P architectures have comparable throughputs, while the UN architecture has the worst throughput.

TABLE IV
HARDWARE COMPLEXITY AND THROUGHPUT FOR SMPL, SMPL-P, UN, UN-P ARCHITECTURES IN THE CASE OF OBLIQUE DT IMPLEMENTATION

Arch	Memory	Mult.	Adders	Throughput (number of instances per second)
UN	$n \times N_a,(n+1)\cdot N_{dt} \times N_c,$ $2\cdot N_{dt} \times \lceil ld(N_{dt}) \rceil, 2\cdot N_{dt} \times \lceil ld(N_{classes}) \rceil$	1	1	$\frac{1}{num\_visited\_nodes \cdot (n+1) \cdot T_{clk}}$
UN-P	$n \times N_a,(n+1)\cdot N_{dt} \times N_c,$ $2\cdot N_{dt} \times \lceil ld(N_{dt}) \rceil, 2\cdot N_{dt} \times \lceil ld(N_{classes}) \rceil$	n	$\lceil 2^{ld(n)+1} \rceil - 1$	$\frac{1}{num\_visited\_nodes \cdot T_{clk}}$
SMpL	$M\cdot(n \times N_a),(n+1)\cdot N_{dt} \times N_c,$ $2\cdot N_{dt} \times \lceil ld(N_{dt}) \rceil, 2\cdot N_{dt} \times \lceil ld(N_{classes}) \rceil$	M	M	$\frac{1}{(n+1) \cdot T_{clk}}$
SMpL-P	$M\cdot(n \times N_a),(n+1)\cdot N_{dt} \times N_c,$ $2\cdot N_{dt} \times \lceil ld(N_{dt}) \rceil, 2\cdot N_{dt} \times \lceil ld(N_{classes}) \rceil$	$M\cdot n$	$M \cdot \left(\lceil 2^{ld(n)+1} \rceil - 1 \right)$	$\frac{1}{T_{clk}}$

V. PERFORMANCE

A. Estimation of savings in hardware complexity of when using SMpL architecture

To illustrate the possible savings in the hardware complexity for the real datasets, five ten-fold cross-validation experiments using the 23 standard UCI datasets [14] were performed.

TABLE VI
EXPECTED SAVINGS WHEN USING SMPL ARCHTIECTURE

Dataset	Architectures [7] and [8]	SMpL Architecture	Savings [%]
ausc	15.58	6.02	61.361
bc	16.86	7.16	57.533
bcw	5.92	4.14	30.068
bsc	10.90	5.70	47.706
car	19.68	7.12	63.821
cmc	108.68	12.94	88.093
ger	24.04	6.82	71.631
gls	14.12	6.14	56.516
hrtc	19.78	6.26	68.352
hrts	6.92	3.98	42.486
ion	4.04	3.30	18.317
liv	19.22	7.48	61.082
lym	4.88	3.38	30.738
page	51.42	9.58	81.369
pid	30.60	8.32	72.810
son	3.10	2.40	22.581
ttt	6.74	4.66	30.861
veh	26.26	9.22	64.890
vow	55.28	8.60	84.443
w21	71.92	12.40	82.759
w40	59.12	10.22	82.713
wpbc	4.64	3.20	31.034
zoo	6.00	3.70	38.333

Following datasets were used: Wisconsin Breast Cancer (bcw), Pima Indians Diabetes (pid), Glass Identification (gls), Vehicle Silhouettes (veh), Vowel Recognition (vow), Statlog Heart Disease (hrts), Australian Credit Approval (ausc), Lymphography Domain (lym), Balance Scale Weight & Distance (bc), Zoo (zoo), Breast Cancer (bsc), Ionosphere (ion), Tic-Tac-Toe Endgame (ttt), Sonar (son), Car Evaluation (car), Contraceptive Method Choice (cmc), German Credit (ger), Heart Disease Cleveland (hrtc), Liver Disorders (liv), Page Blocks Classification (page), Waveform 21 (w21), Waveform 40 (w40), Wisconsin Prognostic Breast Cancer (wpbc).

For every dataset during every cross-validation run, an oblique DT was created. Average number of nodes and average maximum depth of created DTs, over all runs for each dataset are associated with first and second column respectively of Table VI. Third column indicates expected savings in the hardware complexity when using SMpL architecture.

Table VI indicates that SMpL architecture needs 56% fewer node modules than architectures proposed in [7-8] on average. This average saving implies that the hardware realisation of SMpL architecture needs 56% fewer hardware resources in terms of adders and multipliers than the architectures in [7-8]. This would also result in the lower power consumption. From Table VI it can also be seen that the savings increase as the tree size increases.

B. FPGA Implementation Results

Table VII holds the FPGA implementation results for the four proposed architectures in the case of oblique DT implementation using 23 UCI datasets as before. Target device was Virtex5 family FPGA and Xilinx ISE Foundation 12.1.03i software was used to perform the logic synthesis with default synthesis and P&R options. No specific constraints file has been used. For each architecture following data is provided: resource occupation in terms of number of used slices, a minimum clock period and speedup over the software implementation. Software implementation was written in C language and executed using the Xilinx's MicroBlaze [15] embedded microprocessor running at 100 MHz. MicroBlaze is a 32-bit soft processor that can be implemented on any Xilinx FPGA device without any royalties. Xilinx Platform Studio (XPS) 12.1.02i was used to perform the synthesis of the MicroBlaze based

embedded system targeting once more the Virtex5 family FPGA device. Speedup of the hardware DT implementation over the MicroBlaze based implementation was calculated as the ratio of average classification time per instance for MicroBlaze based implementation and average classification time per instance for hardware DT implementation.

TABLE VII
FPGA IMPLEMENTATION RESULTS AND SPEEDUP OVER SOFTWARE IMPLEMENTATION

Dataset	UN			UN-P			SMpL			SMpL-P		
	slices	clk[ns]	speedup	slices	clk[ns]	speedup	slices	clk[ns]	speedup	slices	clk[ns]	speedup
ausc	122	5.063	9.75	43	12.740	58.10	554	6.531	27.28	6140	6.053	441.44
bc	68	4.853	8.72	34	9.524	22.20	426	4.972	26.88	3049	4.265	156.69
bcw	90	3.981	11.95	46	12.040	39.52	314	4.998	19.33	2162	4.184	230.86
bsc	104	4.006	11.88	44	14.016	33.95	485	4.989	31.66	4368	5.000	315.95
car	92	4.812	9.42	43	10.120	31.35	539	6.365	21.50	3769	4.165	230.01
cmc	107	5.548	8.58	35	14.277	33.33	1113	7.066	48.28	9599	7.131	478.43
ger	146	4.389	11.56	60	14.005	90.60	937	6.774	32.14	11184	7.509	724.93
gls	90	4.115	11.56	45	14.016	33.95	480	6.461	30.34	4257	6.958	281.75
hrtc	116	4.248	11.56	38	13.773	49.90	532	6.833	28.16	3526	6.248	431.22
hrts	101	4.323	11.36	43	13.773	49.90	294	4.989	26.27	3071	4.544	403.86
ion	158	4.379	11.73	65	14.116	127.34	439	6.459	18.77	4785	6.250	678.77
liv	92	4.812	9.42	43	10.120	31.35	539	6.365	32.18	3769	4.047	354.30
lym	119	4.278	11.71	56	14.269	66.69	270	4.995	21.76	2982	6.448	320.27
page	107	5.372	8.95	35	11.464	46.12	915	6.928	37.19	7428	6.247	453.63
pid	84	4.930	9.53	40	10.620	39.83	609	6.719	33.22	5144	7.139	281.42
son	295	5.238	9.93	93	14.283	222.10	722	6.548	15.09	5035	6.164	977.80
ttt	78	4.006	11.88	49	14.016	33.95	356	4.974	24.87	3061	4.347	284.60
veh	121	4.135	12.11	54	14.280	66.64	950	6.595	36.61	11183	6.514	704.18
vow	94	5.223	9.40	44	14.260	48.20	857	6.432	43.58	8081	6.665	588.83
w21	111	5.439	9.28	54	14.250	77.91	1410	7.348	37.71	15871	6.793	897.30
w40	238	5.487	9.40	100	14.268	148.22	1851	7.250	38.21	20632	6.343	1790.40
wpbc	142	3.969	12.93	111	14.786	118.00	467	6.465	17.94	4675	6.666	591.52
zoo	88	4.063	12.29	77	14.235	63.14	361	4.990	25.82	3646	5.875	394.70

VI. SUMMARY AND DISCUSSIONS

In this paper several architectures for the hardware realisation of arbitrary DTs (axis-parallel, oblique and non-linear), based on a universal node (UN) or sequence of universal nodes (SMpL), have been proposed. Depending on the way in which instance positions are calculated, four different architectures were presented: SMpL, SMpL-P, UN and UN-P.

A series of experiments have been conducted to compare SMpL architecture with the architectures proposed in [7-8]. Experiments with 23 UCI datasets suggest that SMpL architecture on average needs 56% fewer node modules when compared with the previously proposed architectures.

REFERENCES

[1] C. Bishop, "Neural networks for pattern recognition", Oxford University Press, 1995.

[2] L.Rokach, and O. Maimon, "Top-down induction of decision trees – a survey", *IEEE Trans. on Systems, Man and Cybernetics*, vol. 35, no. 4, November 2005, pp. 476-487.

[3] V. Vapnik, "Statistical learning theory", New York, Wiley, 1998.

[4] D. C. Hendry, A. A. Duncan, and N. Lightowler, "IP core implementation of a self-organizing neural network", *IEEE Trans. on Neural Networks*, vol. 14, no. 5, September 2003, pp. 1085-1096.

[5] S. Himavathi, D. Anitha, and A. Muthuramalingam, "Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization", *IEEE Trans. on Neural Networks*, vol. 18, no. 3, May 2007, pp. 880-888.

[6] D. Anguita, S. Pischiutta, S. Ridella, and D. Sterpi, "Feed-forward support vector machine without multipliers", *IEEE Trans. on Neural Networks*, vol. 17, no. 5, September 2006, pp. 1328-1331.

[7] A. Bermak, D. Martinez, "A compact 3D VLSI classifier using bagging threshold network ensembles", *IEEE Trans. on Neural Networks*, vol. 14, no. 5, September 2003, pp. 1097-1109.

[8] S. Lopez-Estrada, and R. Cumplido, "Decision tree based FPGA architecture for texture sea state classification", *Reconfigurable Computing and FPGA's ReConFig 2006 IEEE International Conference,* September 2006, pp. 1-7.

[9] L. Breiman, J. H. Freidman, R. A. Olshen, and C. J. Stone, "Classification and regression trees", Pacific Grove, CA, Wadsworth and Brooks, 1984.

[10] J. R. Quinlan, "C4.5: programs for machine learning", Morgan Kaufmann Publishers, 1993.

[11] S. K. Murthy, "On growing better decision trees from data", PhD thesis, University of Maryland, College Park, 1997.

[12] A. Ittner, and M. Schlosser, "Non-linear decision trees", *Proceedings of the 13th International Conference on Machine Learning*, 1996.

[13] R. Struharik, and L. Novak, "Evolving oblique and non-linear decision trees", *Internal Report*, FTN 2006.

[14] C. L. Blake, and C. J. Merz, UCI repository of machine learning databases. [Online]. Available: http://archive.ics.uci.edu/ml/.

[15] "MicroBlaze processor reference guide", Xilinx, 2007. [Online]. Available: http://www.xilinx.com/products/design_resources/proc_central/microblaze.htm