LOW COMPLEXITY SIGNAL PROCESSING USING TREE CLASSIFIERS *

Paul D. Fiore[1], Richard J. Barron[2]

[1]*Sanders, A Lockheed Martin Company, Nashua, NH 03061*
[2]*Massachusetts Institute of Technology, Cambridge, MA 02139*

ABSTRACT

Common signal processing operations can be performed efficiently through the use of tree classifiers. Savings in computational complexity are achieved by exploiting problem specifications, which allows the use of approximation and replacement of arithmetic by precomputation. Precomputed results are stored in a tree memory structure that is implemented in VLSI, field-programmable gate array (FPGA), or RAM. In this paper we show that tree classifiers can be the basis for efficient, low-power implementations of several important signal processing operations, including correlation detection and function approximation.

1. INTRODUCTION

Many signal processing operations can be simplified if we consider the accuracy requirements of the resulting outputs. If less accuracy is required, we have the possibility of simplifying the computations while still meeting specifications. A variety of signal processing operations can be viewed as performing analysis on signal vectors residing in a highly structured signal space. The knowledge of the signal space structure lends the problem to low-complexity, non-arithmetic tree-classifier solutions.

Common signal processing operations can be performed efficiently through the use of tree classifiers, which use input feature vectors to divide the feature space into multiple regions. Within each region, a single hypothesis or input class is assumed. With this viewpoint, the tree classifier is seen merely to convert a feature vector into class number. Tree classifiers are distinguished mainly by the set of allowable boundaries used to separate the regions. For example linear and quadratic boundaries have been well studied [1]. We will be interested in cases where the boundaries are given by hyperplanes parallel to the feature space axis.

To illustrate the utility of a tree classifier, an example of acoustic gunshot signature detection is presented. We show that the tree classifier detector requires 1-2 orders of magnitude less energy than conventional techniques.

We have also experimented with estimation algorithms and function approximation. For these tasks, hybrid architectures consisting of both a tree classifier and conventional computation are appropriate. In this paper, we show an example of arctangent calculation. A tree classifier divides the input space into rectangles. Each rectangle has an associated bilinear approximation which is used to compute the arctangent.

*PREPARED THROUGH COLLABORATIVE PARTICIPATION IN THE ADVANCED SENSORS CONSORTIUM SPONSORED BY THE U.S. ARMY RESEARCH LABORATORY UNDER COOPERATIVE AGREEMENT DAAL01-96-2-0001

2. FPGA IMPLEMENTATION OF A TREE CLASSIFIER DETECTOR

We wish to detect the occurrence of a template signal corrupted by Gaussian noise. Let the template consist of the vector $\mathbf{a} = [a_1, \dots a_n]^T$ and let the data within the correlation window be denoted by $\mathbf{x} = [x_1, \dots x_n]^T$. Using a correlator for detection, the correlation output is

$$\mathrm{C} = \mathbf{a}^T \mathbf{x} \tag{1}$$

A detection is declared when the correlator output is larger than a threshold. We can define a decision function $H(\mathbf{x}; \mathbf{a})$ such that

$$H(\mathbf{x}; \mathbf{a}) = \begin{cases} 1 \text{ if } \mathrm{C} \geq \eta \\ 0 \text{ if } \mathrm{C} < \eta \end{cases} \tag{2}$$

In this instance, the actual value of the correlation C is not of interest, only its relationship to the threshold η. This scenario is described by following two hypotheses:

$$H_0 : \mathbf{x} = \mathbf{v} \tag{3}$$
$$H_1 : \mathbf{x} = \mathbf{a} + \mathbf{v} \tag{4}$$

The zero-mean Gaussian noise vector $\mathbf{v}$ has covariance matrix Λ. If $\Lambda = \sigma_v^2 \mathbf{I}$ where $\mathbf{I}$ is the identity matrix, then a likelihood ratio test decision rule is equivalent to a correlation receiver operating according to (2).

One generally performs a likelihood ratio test, which amounts to comparing some function of the input data to a fixed decision threshold [2]. The decision threshold η is chosen to minimize a certain average cost for making the decision. For any detection rule, there exist a probability of false alarm P_F and a probability of detection P_D. P_F and P_D are dependent on the conditional densities of the measured data $\mathbf{x}$, and the rule by which the decision is made.

As an alternative, we can use a binary tree classifier. The classifier examines one bit of the input vector $\mathbf{x}$ at a time. Based on the value of the bit, the classifier may decide to stop examining the remaining bits and classify the input, or it may decide to continue examining bits. In the later case, the value of the bit just examined determines which of two "branches" that the classifier will take.

This method will be efficient if the tree is "sparse". Sparseness occurs when many decisions can be made early in the tree expansion. Nodes at which decisions can be made are not subsequently expanded, thus limiting the number of descendents.

Several straightforward RAM-based implementations can easily be envisioned for the sparse tree structure. If one has available general purpose logic resources rather than large amounts of memory resources, then it is more economical to lay the sparse tree out in parallel. The data would now "snake" down through the tree from parent to child. The appropriate child would be chosen according to the data present at the parent node.

While a RAM-based implementation can handle larger trees than any single FPGA device, the FPGA implementation has the advantage that it is extremely low in power dissipation. This is because the only logic transitions that occur are on the active tree descent path. Nodes not on the path never receive a valid data signal, so they do not generate transitions. We show an example where the FPGA energy requirements are several orders of magnitude below what would be required for either a RAM-based implementation or even a conventional DSP microprocessor software implementation.

2.1. Designing the Tree

We now describe a method for designing the sparse tree. Here we take the approach that the user specifies the desired P_F and P_D, the template $\mathbf{a}$, the number of bits in the data b, and the noise level σ_v^2. For simplicity, we will assume a fixed bit examination order, which can be efficiently implemented in the FPGA. Variable examination orders allow for smaller trees, at the expense of more complicated node hardware.

In our fixed examination order, bits of the input vector are examined according to their contribution in reducing the uncertainty in the output correlation. For example, the MSB of the input word corresponding to the template word of maximum absolute value is examined first. While the bit sequence will in general be interspersed among many input words, the sequence will have the property that a bit will not be examined unless all the bits in the same word of larger weight have been previously examined.

The algorithm proceeds in an iterative manner. At each stage, a complete sparse tree is designed, but it may not meet the P_F and P_D specification. The algorithm first calculates the P_F and P_D that the current tree will achieve. These quantities can be calculated by considering each leaf node individually. For a particular leaf node, the sequence of bits that were examined and their actual values are known. The values of many other bits are not known. This knowledge enables us to confine the actual values to a hyperrectangularly shaped region in the space whose axes are $[x_1, \ldots x_n]^T$. The probability that we arrive at this leaf node under each hypothesis H_0 and H_1 is simply the volume under the conditional probability density functions (PDFs) $p_X(x|H_0)$ and $p_X(x|H_1)$ in the hyperrectangular region. This volume is easy to calculate because of the assumption of white noise. In this case, the multidimensional conditional PDFs decompose into products of univariate Gaussian PDFs.

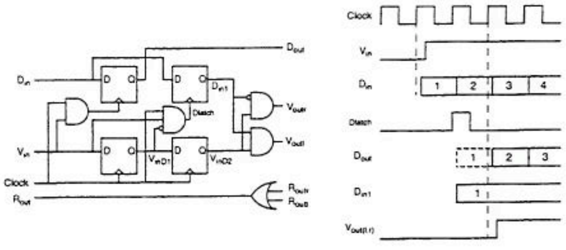

Figure 1. Basic Node Schematic and Timing

Now we have the probabilities of arriving at a particular leaf node under H_0 and H_1. In our algorithm, a tentative decision is made by choosing the hypothesis with the larger probability. These calculations are performed for all leaf nodes. Since the leaf nodes represent mutually exclusive events, it is straightforward to calculated the overall P_F and P_D of the current sparse tree. If these values do not meet the system specifications, one of the current leaf nodes must be expanded. By this we mean that a leaf node must be transformed into a parent node, with two children allocated. A particular bit of the input must also be designated for examination (for fixed examination orders, this bit is predetermined).

To choose which leaf to expand, we employ a simple greedy algorithm. The algorithm chooses the leaf node that contributes the most to the probability of error, relative to the desired P_F and probability of miss, P_M ($= 1 - P_D$). Once this leaf is identified, it is expanded.

2.2. Tree Implementation

For our FPGA implementation, each node in the tree represents a fundamental module. This allows for the implementation of any given tree through replication and interconnection of modular nodes. The logic contained in a node is illustrated in figure 1. Two flip-flops (Vind$_1$ and Vind$_2$) are used to generate the appropriate timing for node control signals. Another is used to pipeline the data, and the remaining flip-flop (Din$_1$) is used to store the tree branching condition. The OR gate is used to as a means of passing the classification result from either Routr or Routl to the root node of the tree via Rout.

Leaf nodes in the design are simpler than internal nodes. A leaf node must simply route its Vin signal back to the root node. This will only occur if the leaf node corresponds to the H_1 hypothesis. In our implementation, this is accomplished by routing a leaf node's asserted Vin back to the parent node's Routr or Routl.

A complete tree is constructed by starting with a single node as the root of the tree with some external logic piping the data to Din, an enable signal to Vin, and a clocking signal to Clock. Child nodes are connected to their parents by linking the datastreams (Din to Dout), and enable signals (Voutr/Voutl to Vin). Results are passed back to the parent by linking Rout to either Routr or Routl. Leaf nodes will directly drive Routr or Routl from their Vin signal (which is generated by the parent's Voutr or Voutl).

2.3. Placement Algorithm

Initially we looked at adapting some of the work done in optimal layout of full binary trees into VLSI arrays [3]. However, it was unclear how these procedures could be modified for sparse trees, so we developed an ad hoc algorithm for this purpose.

To map the sparse tree into the FPGA, a simple greedy placement strategy was used. The algorithm first identifies the shortest tree path from the root to a leaf node. All the nodes along this path are placed first. Then the next shortest path from the root to an unplaced leaf node is determined, and every node along this path is placed next. This sequence proceeds until all nodes are placed. Within a path, the nodes are placed starting from the one closest to the root node and continuing until the leaf node from that path is placed.

Once it is determined that a particular node is to be placed next, its location must next be determined. The algorithm starts with the location of the node's parent node. From the parent node location, the closest free locations in the horizontal and vertical directions are located (there are a maximum of four). If there are multiple nodes, then the one closest to the tree's root node is selected. The determination of distance to the root node is weighted by the basic node aspect ratio (which is 1x2 in our design). By using a weighted distance to the root node, the tree remains clustered around the root node, and the tree will tend to remain square.

The placement algorithm will be successful if most of the time it can place a child node adjacent to the parent node. Nodes placed progressively farther away from their parent will require increased routing resources, which can quickly make the design unroutable or too slow. By placing short paths first, more room is left for the longer paths which have potentially more branches.

2.4. Design Example

Here we use a particular template as an example. Shown in figure 2, the template consists of 64 samples of the acoustic signature of an M16 rifle. For this design we desired a $P_F = 10^{-4}$, $P_M = 10^{-5}$, and $\sigma_v^2 = 256$. This corresponds to a signal to noise ratio of approximately 10dB. This design required 325 internal nodes, and 326

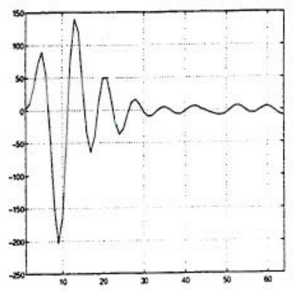

Figure 2. 64-point Template of M16 Signature

leaf nodes. Figure 3 shows the layout generated by the above algorithm.

In this particular layout, leaf nodes are not assigned a node location, because their logic is absorbed into their parent's output logic. A line segment represents a connection from a parent to child node. In some instances, a parent and child are not adjacent, so the line segment connecting them in the figure is simply drawn straight through any intervening nodes. The figure is scaled to the correct node aspect ratio. To calculate routability of the layout, let the route length be denoted by the maximum difference in rows or columns between parent and child node. An ideal layout would thus have all routes of length one. The layout in figure 3 has an average route length of 1.074, which indicates that very few nodes could not be placed adjacent to their parent. The average route length increases when the tree size approaches the limit of nodes imposed by the finite size of the FPGA.

We have written a set of MATLAB routines that design the tree, determine node placement within the FPGA, and generate the netlist and placement constraint files. The design is then imported into the Viewlogic and Xilinx toolsets, where operations such as routing, back-annotation, and simulation can take place.

Figure 3 also shows a portion of the actual placement and rats nest of the tree in an FPGA. The rats nest does not look like a typical one because of the successful placement algorithm.

Let us now consider the power dissipation of the design. Each active node generates a maximum of seven transitions. The average number of levels that are descended under the H_0 hypothesis for this tree is approximately two, for an approximate power dissipation of 3mW/MHz using a Xilinx XC4000 FPGA. The global clock dissipates 40mW/MHz, for a grand total of 43mW/MHz/classification, which translates into 43nJ/classification. In contrast, the data sheet of a very power efficient DSP microprocessor [4] gives a specification of 303.6mW/66MHz/instruction. Since the template is 64 samples in length, the energy required is 64·303.6mW/66MHz/classification, which translates into 300nJ/classification. This is approximately 7 times more energy on average than the tree classifier approach. Elimination of the global clock using an asynchronous logic design would increase this ratio by another order of magnitude.

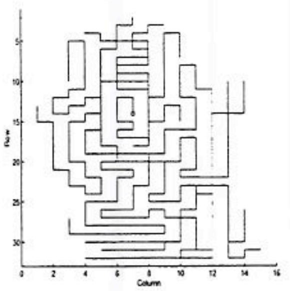
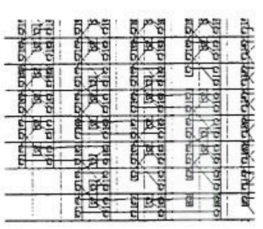

Figure 3. Tree Layout and Floorplanner Placement Closeup

3. ESTIMATION AND FUNCTION APPROXIMATION

The preceding detection problem had an efficient sparse tree solution partly due to the fact that the variety of possible outputs was restricted to only two choices. Estimation problems, generally having a multitude of possible outputs, will lead to more complicated boundaries between the output "classes". Thus, it seems unlikely that an estimator based solely on a tree classifier will be an efficient implementation.

We are led to consider hybrid combinations of a tree classifier and more conventional computational methods. Since many estimators are simply of the form of a function evaluation, let us consider an example where a sparse tree is used to select among different approximations of the desired function. The approximation will change based upon the hyperrectangle in which the input vector is located.

We will use the two-input arctangent function as an example. Arctangents can be calculated a variety of ways, including CORDIC rotations [5] and lookup tables. Here, a sparse binary tree classifier divides the input space into rectangles. Each rectangle has an associated affine approximation which is used to compute the arctangent. That is, within rectangle r, the approximation is calculated via

$$Atan_2(x, y) \approx a_r x + b_r y + c_r \qquad (5)$$

Within each rectangle, the coefficients are chosen to minimize the mean square error of the approximation. Rectangles are subdivided until the total error is below the desired threshold. Figure 4 shows an example where the input words consisted of five bits and the output mean-square error was 4.8×10^{-4}. A total of 41 leaf nodes were used.

There are several opportunities to further optimize this design. At each leaf node, some of the bits of x and y are known and some are not. Let (x_k, y_k) denote the values of the known portions and (x_u, y_u) the unknown portions. Then

$$a_r x + b_r y + c_r = (a_r x_k + b_r y_k + c_r) + (a_r x_u + b_r y_u) \quad (6)$$

Now the values $a_r x_k + b_r y_k + c_r$, a_r, and b_r are precomputed and stored for each leaf node. The multiplications that must be performed in real-time are now simpler than if the entire bits strings of (x, y) were used. Further improvement can be made with the addition of normalization logic and octant reflection identities before the data is input to the tree.

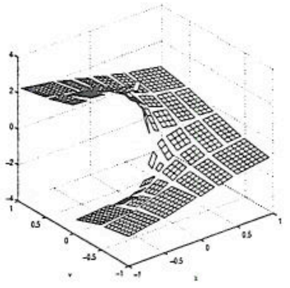

Figure 4. Bilinear Arctangent Approximation

4. CONCLUSIONS AND FUTURE WORK

There are several areas that remain to be explored. Variations in the basic node architecture are certainly possible. One variation would be to remove the global clocking structure, which would reduce power dissipation. Another would be to allow for variable bit examination orders. For estimation problems, efficient methods of storing the estimate could be examined. All of these involve a tradeoff between tree size and node complexity.

Trees with sizes greater than that which can be accommodated by a single FPGA device will need to be partitioned into multiple devices. We are currently researching the issues of designing the trees subject to partition size and pinout constraints.

Because of the locality of activity from one node to the next, the sparse tree structure is ideal for implementation in an adaptive computing environment. These structures are similar to conventional FPGAs, but allow for some degree of reprogrammability on a single clock cycle basis. When node activations hit the boundary of the array, the array can be reprogrammed to the correct subtree. Thus, the activations wander from boundary to boundary until a decision is reached.

REFERENCES

[1] Richard O. Duda and Peter E. Hart. *Pattern Classification and Scene Analysis.* Wiley Interscience, 1973.

[2] Anthony D. Whalen. *Detection of Signals in Noise.* Academic Press, 1971.

[3] Dan Gordon. Efficient embedding of binary trees in VLSI arrays. *IEEE Trans. on Computers*, 36(9), September 1987.

[4] Motorola Inc. *DSP56302 Data Sheet.* 1996.

[5] Yu Hen Hu. CORDIC-based VLSI architectures for digital signal processing. *IEEE Signal Processing Magazine*, July 1992.