Minimal Delay Interconnect Design Using Alphabetic Trees

Ashok Vittal and Malgorzata Marek-Sadowska

Department of Electrical and Computer Engineering,

University of California

Santa Barbara, CA 93106

Abstract - We propose a new algorithm for the performance-driven interconnect design problem, based on alphabetic trees. The interconnect topology is determined in a global manner and does not greedily add edges as in conventional approaches. The algorithm can handle cases where the sink capacitances are different. Good results are obtained while running two to sixty times faster than three existing algorithms on practical instances.

I. Introduction*

Interconnect delay has become a significant fraction of the clock cycle time in high-performance digital systems due to the scaling down of feature sizes in integrated circuits and increasing on-chip interconnect line lengths [1]. This has made performance-driven layout an important issue in the design of these systems. Motivated by this, timing-driven placement algorithms have been proposed in [2] and [3]. These methods try to place cells on critical paths close to each other. While timing-driven placement alleviates some of the problems, it has to be complemented by good performance-driven routing to achieve optimized layouts.

Early work on performance-driven routing abstracted the problem into a purely geometric form. The interconnect was treated as a lumped capacitance so that the objective was to minimize the total wire length, i.e., the problem was reduced to one of finding the minimum Steiner tree under the Manhattan distance metric. Two assumptions were implicit in this formulation - the interconnect line lengths were much less than the wavelengths of the signals being propagated so that inductance effects were absent and interconnect resistance was negligible in comparison with the driver resistance, making a lumped capacitance model adequate. The decreasing driver resistance to interconnect resistance ratio as technology scales down has made this formulation less appropriate. The larger frequencies at which circuits work have also meant that transmission line effects have to be taken into account for smaller interconnect lengths. Besides, sink capacitances can vary considerably due to transistor sizing optimizations. The topology that yields the smallest delays is affected by the load capacitances, so that any geometric abstraction completely misses this dimension of the problem.

Good models for interconnect delay calculation are necessary if an interconnect design algorithm is to obtain satisfactory results. The Elmore delay model [4] has come in handy for this task. It has been shown to be a high-fidelity estimator of the delay of routing trees [5],[6]. A careful evaluation of the Elmore delay expression led to the A-tree algorithm in [7]. However, this algorithm was a geometric abstraction. The algorithm that has reported the best results so far is the Elmore routing tree algorithm [8]. This algorithm constructs a routing tree, adding edges greedily. The Steiner version (SERT) of this algorithm runs in $O(n^4)$ time and the basic algorithm (ERT) runs in $O(n^3)$ time where n is the number of sinks. The trees that the SERT algorithm returns have been shown to be near-optimal in [5].

Our contribution arises from viewing the performance-driven routing problem in the context of design flow and optimizing the Elmore delay in a global fashion. Place and route systems are notorious for large computation time consumption. Many expensive layout-extract-simulate iterations may be required to satisfy performance constraints. The interconnect design algorithm should not take too much time. On the other hand, it should give results that are close to optimal. A greedy algorithm spends a large amount of time exploring parts of the design space that could not possibly be optimal. A global approach could maintain the quality of the results and yet not consume too much time. Both the generic routing algorithms proposed in [8] are greedy. While the results they get are close to optimal, the computation time grows very rapidly with the number of sinks. We note that nets with a large number of pins are not uncommon. The state bit outputs of a finite state machine, represented by a strongly-connected state transition graph, fans out to a large number of next state logic inputs. Multi-level logic synthesis techniques increase logic sharing and can lead to large fanouts. At the system level, all global control signals are typically large nets with large fanouts. In this paper we propose an algorithm that solves the problem in a global manner, giving results that are just as good as those produced by the two greedy algorithms while running in $\mathbf{O(n^2)}$ time.

The algorithm we propose handles variable sink capacitances naturally. Transistors in present day integrated circuits

*. This work was supported in part by the Advanced Research Projects Agency under contract DABT63-93-C-0039, the National Science Foundation under Grant MIP 9117328 and in part by AT&T Bell Laboratories and Digital Equipment Corporation through the California MICRO program.

are sized to optimize performance. The gate capacitances of transistors can therefore vary over a wide range. Geometric abstractions like [7] return the same tree regardless of sink capacitances. Hence, the results may be far from optimal.

II. Problem Formulation

We wish to minimize the weighted sum of sink delays given the source and sink positions. Ideally, the delay should be computed using an extraction program followed by a simulation run, on SPICE for example. However, this would be very expensive in terms of computation time. We therefore seek tractable methods to estimate the delay. Numerical methods, like Asymptotic Waveform Evaluation [9], can be used to speed up the computation. This could then be used in an optimization loop. Such a method would use up a large amount of time exploring parts of the design space which could not possibly be optimal. We therefore use an analytical model for the interconnect delay which is not only fairly accurate but also allows an intuitive understanding of the problem. Besides, it allows the delays to all the nodes in a routing tree to be calculated in linear time [10]. The Elmore delay has been shown to be a high-fidelity estimator of interconnect delay [5],[6]. Thus we formulate the problem as follows.

Given a set P = $\{p_1,p_2,...,p_n\}$ of sinks on the Manhattan plane with each sink having an associated position Z_i, capacitance C_i & weight W_i, a source with a position Z_0 & driver resistance R_d, and unit grid resistance R_o & capacitance C_o, construct a tree which minimizes

$$\sum_{i=1}^{n} W_i T_i$$

where T_i is the Elmore delay to the i^{th} sink and is given by

$$T_i = \sum_{k \in P \cup S} R_{ki} C_k$$

where R_{ki} is the resistance of the path that is common to the path from the root to the node i and the path from the root to the node k of the lumped equivalent RC model, C_k is the total capacitance at a node k in the tree and S is the set of internal nodes of the tree.

The above formulation has implicitly replaced each segment of interconnect by a lumped pi-equivalent RC section. We note that the above definition of delay is not affected by the granularity of the model, that is, the Elmore delay remains the same even if an interconnect segment is represented by a cascade of many RC sections. This inherent robustness further justifies the use of the Elmore delay as an estimator.

III. Alphabetic Tree-Based Algorithm

The Elmore delay to sink p_i can be written(as in [7]) as

$$T_i = R_d C_o L_t + R_d \sum_{k \in P} C_k + R_o \sum_{k \in P} C_k L_{ki} + R_o C_o \sum_{k \in S} L_k L_{ki}$$

where L_t denotes the total wire length of the tree, L_k is the length of the path to node k from the root and L_{ki} is the length that is common to the paths to the nodes k and i from the root. As before, S is the set of internal nodes of the tree, P is the set of sink nodes. R_d is the driver resistance, C_k is the node capacitance at node k and C_o & R_o are the capacitance & resistance per unit grid.

Of the 4 terms in this expression, the second is a constant and is topology-independent. From the third and fourth terms, we see that the length term L_{ki} has to be minimized when the capacitance term that multiplies it is large. The topology of a tree should be selected such that nodes with large sub-tree capacitance are topologically closer to the root. However, the total wire length term should also be kept small so that the first term does not become too large. These observations are the basis of our algorithm. We now look at the alphabetic tree problem and motivate its use in solving the interconnect design problem.

The alphabetic tree problem is stated as:

Given an ordered set of weights $\{w_i\}$ find a binary tree such that the weighted sum of path lengths to the leaves is minimum among all such trees and the left to right order of the leaves of the tree is maintained. A binary tree is defined as a tree where each node has either 2 children or no children (note that nodes with one child are not allowed). The path length of a leaf is defined as the number of edges on the path from the root to that leaf.

We note the strong correspondence between the performance-driven routing problem and the alphabetic tree problem. In particular, we note that the placement imposes a geometric order on the sinks. By keeping the tree alphabetic we bound the total wire length of the tree and thus the first term of the Elmore delay expression. By using appropriate weights we keep the last two terms of the Elmore delay expression small. In the alphabetic tree problem large weight nodes are close to the root in the solution. Similarly, for small delays, large capacitance nodes must be topologically close to the root.

Consider the example shown in Figure 1. The left to right ordering and the weights of nodes have been specified in the input. The optimum tree has a weighted path of length 53 and is shown on the right. The node with the largest weight is closer to the root, as expected.

The number of alphabetic trees grows exponentially with n and makes exhaustive enumeration impractical. The problem is similar to that of constructing optimal prefix codes [11], with the additional constraint of the left-to-right ordering. While the optimal prefix code construction problem can be solved by a greedy algorithm (the Huffman coding algorithm[12]) which constructs a tree representing the

code in a bottom-up manner, this approach does not work for the alphabetic tree problem. The example in Figure 1 also illustrates why a one-pass bottom-up merging of nodes as in the Huffman coding algorithm is not sufficient to solve the problem optimally. Such an algorithm would merge nodes of weight 2 and 3 in the first step. The tree generated would not be optimal whatever the other two merges may be.

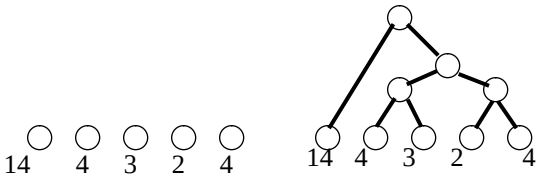

Fig. 1. An alphabetic tree problem instance and its solution

Optimum path length = 14+3(4+3+2+4) = 53

This problem can be solved in O(n log n) time using the Hu-Tucker algorithm [13], [14]. The alphabetic tree algorithm finds applications in routability-driven fanout optimization [15] and binary encoding problems [14]. The Hu-Tucker algorithm, which we use, has 3 stages - combination, level assignment and recombination as in [14]. The algorithm is outlined below.

Alphabetic tree algorithm

Input: Totally ordered set Q={q1,q2,...,qi,...qn}
Output: Minimum alphabetic tree T

Repeat n-1 times {

Combination : Combine nodes i and j which yield the smallest weight when combined and are either adjacent or have only internal nodes in between them }

Level assignment: Assign levels to leaves using the topology returned by the combination phase. The level is just the distance from the root.

Recombination: Discard the initial tree and use the leaf level assignment to form the tree again. The highest level nodes are combined in a bottom-up manner to get the optimal alphabetic tree T.

In our heuristic the order is obtained by sorting the nodes with angle subtended by the ray connecting the point and the source with the X-axis as the key, i.e., we use a circular ordering of the sinks. The circular ordering ensures that the output tree is planar. The weights are just the sink capacitances. The combination of node weights includes the capacitance of the wiring. Thus when two sub-trees are combined, the weight of the new sub-tree is the sum of the two sub-tree capacitances and the wiring capacitance. The wiring capacitance is that of the shortest segment that connects the roots of the two sub-trees. It is important to keep the path from the source to any sink as small as possible because this affects the system time constant. Our heuristic always returns a tree in which the path to any sink from the source is a shortest path. In other words, the path from the source to any sink is a monotonic path. This is done by assigning the position of the root of the new sub-tree when combining two sub-trees as shown in Figure 2 below.

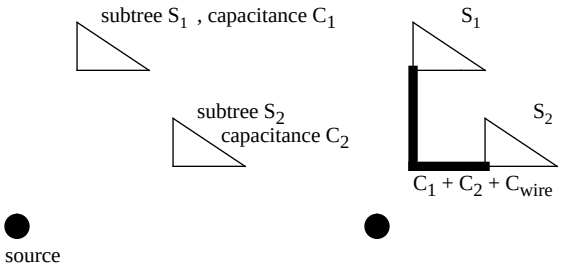

Fig. 2. Merging subtrees S_1 and S_2 : The root of the new subtree has the smallest X- and Y- coordinates of any of its leaves. This makes the path from any sink to the source a shortest path.

The binary tree restriction of alphabetic trees is not too severe. When subtrees are combined, the position of internal nodes is fixed as in Figure 2. Some internal nodes may therefore degenerate to higher degrees.

Having obtained a good topology, we check if further improvement is possible using local descent optimization, i.e., we post process the binary alphabetic tree so that internal nodes can migrate towards the root. This is done by checking, going bottom up, if moving an internal node to the position of its parent will improve average delay and make the change if this is the case.

Algorithm ALPHA

Input: Technology, source & sink positions
Output: Minimal delay topology

Generate alphabetic tree T, using circular ordering

For each internal node X, in bottom up order

Move X to the position of its parent if this move improves average delay.

The time complexity of generating the alphabetic tree is O(n logn). The optimization, however takes $O(n^2)$ time as n-1 internal nodes can move up and each decision involves computing the Elmore delay, which can be done in linear time[10]. The algorithm, therefore, has time complexity of $O(n^2)$.

We note another property of the trees generated by ALPHA which trees generated by SERT and ERT may not have.

Property 1

The sink capacitance contribution to the system time constant T_p of the tree generated by ALPHA is the minimum possible for routing trees in the Manhattan plane and is given by

$$T_p = \sum_{\forall (k \in P)} R_{kk} C_k$$

$$= R_o \sum_{\forall (k \in P)} C_k (X_k + Y_k)$$

where X_k and Y_k are the X and Y coordinates of sink k.

This is true because of the way in which sub-trees are combined. As the path to each sink is monotonic, R_{kk} (the resistance of the path from the source to the sink) is the resistance per unit length multiplied by the distance of the sink from the root.

This property of minimum system time constant is useful since voltage bounds as a function of time [10] at any sink have this as the time constant. With all else being equal, a difference in T_p could lead to a large difference in the actual delay obtained.

IV. Results and Conclusions

We have implemented our algorithm in C on a DEC 3100 workstation and compared our results with that of the SERT[8], ERT[8] and 1-Steiner algorithms[16], when run using the same compiler and operating system. The SERT and ERT algorithms have been reported to perform better than the A-tree[7], the bounded-radius bounded-cost[17] and the AHHK[18] algorithms. The 1-Steiner algorithm gives us an estimate of the minimum wirelength required. We compared the average delays and the average wire lengths of the trees generated by these algorithms for sets of 1000 randomly generated nets for two different IC technologies (IC1 and IC2). The resistance and capacitance per unit length of IC1 were extracted using the OptEMVLSI electromagnetic interconnect analysis package[19] for MOSIS2NC technology with 2 micron metal1 lines. IC2 parameters are for a technology with smaller line-widths so that the resistance per unit length is larger. The capacitance per unit length has not changed because of fringing fields[1].

TABLE 1. Technology parameters

Technology	R_o(ohms/ micron)	C_o (pF/cm)	C_l (fF)	R_d (ohms)
IC1	0.35	0.53	5.0	150
IC2	0.70	0.53	5.0	150

We ran experiments to see how the average Elmore delay to the sinks and the average wirelengths used varied with the number of pins in the net and the physical size of the net, for both technologies. The run time (user time) required by the four algorithms were also compared. The grid size and net size are typical for critical nets in an IC. The results obtained are shown in the tables. All the results are normalized with ALPHA=1. The 95% confidence intervals for all the delay and cost ratios are less than 0.002.

TABLE 2. Average delay comparison (ALPHA=1) - Variation with grid size (5-pin nets)

grid size (mm)	SERT IC1	SERT IC2	ERT IC1	ERT IC2	1-ST IC1	1-ST IC2
0.5	1.45	1.46	1.51	1.49	0.96	1.00
2.0	1.07	1.07	1.09	1.07	1.09	1.21
5.0	1.00	1.00	1.00	0.99	1.23	1.35

TABLE 3. Average delay comparison (ALPHA=1) - Variation with the number of sinks (2mm grid size)

N	SERT IC1	SERT IC2	ERT IC1	ERT IC2	1-ST IC1	1-ST IC2
4	1.07	1.07	1.09	1.07	1.09	1.21
6	1.04	1.03	1.08	1.04	1.11	1.26
8	1.01	1.00	1.06	1.02	1.11	1.28

TABLE 4. Average wirelength comparison (ALPHA=1) for 5-pin nets

grid size (mm)	SERT IC1	SERT IC2	ERT IC1	ERT IC2	1-ST IC1	1-ST IC2
0.5	0.68	0.78	0.80	0.87	0.65	0.66
2.0	0.81	0.95	0.91	1.00	0.68	0.71
5.0	0.95	1.04	1.01	1.07	0.73	0.75

TABLE 5. Run time comparison (ALPHA=1)

N	SERT	ERT	1-Steiner
4	2.4	1.7	12
6	4.4	2.9	29
8	7.9	4.6	60

ALPHA consistently returns small delay topologies over the entire range of technologies, grid size and net size under consideration. We also see that as the minimum Steiner tree approximation becomes invalid, ALPHA begins to return trees whose delay and wirelength are competitive with that of the SERT and ERT algorithms. This is achieved while running 2 to 60 times faster than the other algorithms. Besides, the wirelength gets better as technology scales down (IC2 has smaller feature size).

We now look at some typical examples. The first example shown in Figure 3 illustrates how a greedy algorithm might be led to a local optimum. The second example shown in Figure 4 illustrates the fact that non-monotone paths to

sinks may result in large delays.

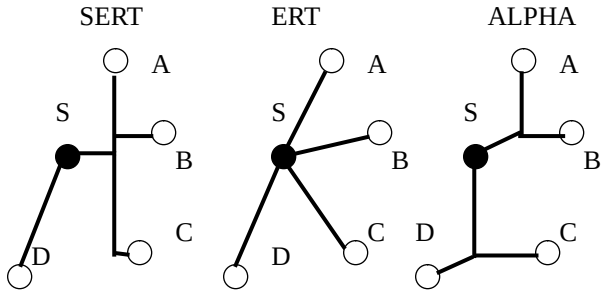

Fig. 3. Pathological case: ERT & SERT: 24,20% worse

In the example shown in Figure 3, SERT adds an edge to A as this is the closest sink to the source. An edge is then added from B to the existing edge. This is followed by the addition of an edge from C to the existing edge to A - a wrong decision. The best SERT can do now is to have a

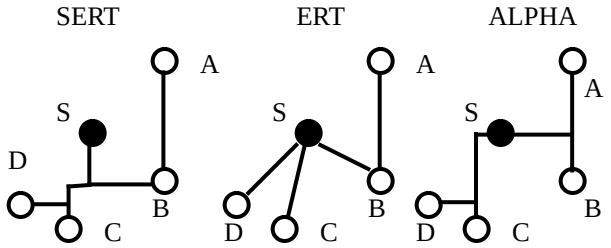

Fig. 4. Non-monotonic paths generated by ERT & SERT

direct connection from D to the source. Note that if in the third step an edge was added to the source instead of adding it to the existing edge (which a greedy algorithm fails to do), the resulting tree would be much better. The fundamental problem with the greedy approach is that it makes ill-informed decisions which may turn out to be bad later. The ERT algorithm constructs a spanning tree greedily and is also far from optimal. The average Elmore delay for Alpha is 20% better than SERT and 24% better than ERT for this instance.

For the input instance shown in Figure 4, SERT and ERT generate trees that have a non-monotonic path to sink A. This results in large average delays. For this example SERT was 40% worse than ALPHA and ERT was 45% worse.

When ALPHA returned large Elmore delays, we found that this could be remedied by changing the alphabetic order. However, the increase in run time may not be worth the incremental improvement in performance.

We conclude that the algorithm we propose can avoid some of the pathological cases for SERT and ERT, the two algorithms that have reported the best results in the literature. Our algorithm runs much faster and can be used in a layout system for performance-driven interconnect design. Our algorithm scales well with net sizes, both with respect to run time and quality of results obtained. The global manner in which the interconnect design is done helps our algorithm to avoid local optima.

The alphabetic tree framework is quite general and could be used to solve other VLSI routing problems, such as power supply net routing. A global router which takes advantage of the hierarchical nature of our heuristic could also be developed.

Acknowledgments

We would like to thank the authors of [8] for the use of their SERT, ERT and 1-Steiner code. We also thank Mr.Kenneth Boese and Professor Andrew Kahng of the University of California, Los Angeles for numerous helpful discussions.

References

[1] H.B. Bakoglu, *Circuits,Interconnections and Packaging for VLSI*, Addison-Wesley, 1990, pp. 195-198.

[2] W.E.Donath, R.J.Norman, B.K.Agrawal, S.E.Bello, S.Y.Han, J.M.Kurtzberg, P.Lowy and R.I.McMillan, “Timing-driven placement using complete path delays”, Proc. DAC, 1990, pp.84-89.

[3] M. Marek-Sadowska and S.Lin, “Timing-driven placement”, Proc. ICCAD, 1989, pp. 94-97.

[4] W.C. Elmore, “The transient response of damped linear networks with particular regard to wideband amplifiers”, Journal of Applied Physics, Vol.19, 1948, pp. 55-63.

[5] K.D.Boese, A.B.Kahng, B.A.McCoy and G.Robins, “Fidelity and near-optimality of Elmore-based routing constructions”, to appear, Intl. Conf. on Computer Design,1993.

[6] S.Kim, R.M.Owens and M.J.Irwin, “Experiments with a performance-driven module generator”, Proc.DAC,1992, pp. 687-690.

[7] K.S.Leung, J.Cong and D.Zhou, “Performance-driven interconnect design using a distributed RC delay model”, Proc. DAC, 1993, pp.606-611.

[8] K.D.Boese, A.B.Kahng and G.Robins, “High-performance routing trees with identified critical sinks”, Proc. DAC 1993, pp.182-187.

[9] L.T. Pillage and R.A. Rohrer, “Asymptotic waveform evaluation for timing analysis”, IEEE Trans. on CAD 9(4), April 1990, pp.352-366.

[10] J.Rubinstein, P.Penfield and M.A.Horowitz, “Signal delay in RC tree networks”, IEEE trans. on CAD 2(3), 1983, pp. 202-211.

[11] T.H.Cormen, C.E.Leiserson and R.A.Rivest, *Introduction to Algorithms*, 1990, MIT Press and McGraw-Hill Book Co., pp.347-345.

[12] D.A.Huffman, “A method for the construction of minimum-redundancy codes”, Proc. IRE, 40(9), 1952, pp.1098-1101.

[13] T.C. Hu and A.C. Tucker, “Optimal computer search trees and variable length alphabetic codes”, SIAM Journal of Applied Mathematics, 21(4), 1971, pp. 514-532.

[14] D.Knuth, *Sorting and Searching*, Vol. 3 of *The Art of Computer Programming*, Addison-Wesley, 1973, pp.439-445.

[15] H.Vaishnav and M.Pedram, “Routability-driven fanout optimization”, Proc. DAC, 1993, pp.230-235.

[16] A. Kahng and G.Robins, “A new class of Steiner tree heuristics with good performance: the iterated 1-Steiner approach”, Proc.ICCAD, 1990, pp.428-431.

[17] J.Cong, A.B.Kahng, G.Robins, M.Sarrafzadeh and C.K.Wong, “Provably good performance-driven global routing”, IEEE Trans. on CAD,11(6), June 1992, pp.739-752.

[18] C.J.Alpert, T.C.Hu, J.H.Huang and A.B.Kahng, “A direct combination of the Prim and Dijkstra constructions for improved performance-driven global routing”, Proc.ISCAS, 1993, pp.1869-1872.

[19] OptEM Engineering Inc., OptEM VLSI User’s Manual, Interconnect Optimization and Electromagnetic Analysis, Version 2.2, 1993.