A Simultaneous Routing Tree Construction and Fanout Optimization Algorithm*

Amir H. Salek, Jinan Lou, Massoud Pedram
Department of Electrical Engineering - Systems
University of Southern California
Los Angeles, CA 90089
{amir, jlou, massoud}@zugros.usc.edu

ABSTRACT - This paper presents an optimal algorithm for solving the problem of simultaneous fanout optimization and routing tree construction for an ordered set of critical sinks. The algorithm, which is based on dynamic programming, generates a rectilinear Steiner tree routing solution containing appropriately sized and placed buffers. The resulting solution, which inherits the topology of LT-Trees and the detailed structure of P-Trees, maximizes the signal required time at the driver of the given set of sinks. Experimental results on benchmark circuits demonstrate the effectiveness of this simultaneous approach compared to the sequential methods.

1. INTRODUCTION

The current deep-submicron (DSM) process technologies have increased the contribution of the interconnect delay to the total path delay in digital circuits. At the same time, the existing design flows and tools have had limited, and only marginal, success in incorporating interconnect planning and optimization early in the design process. This situation has forced IC designers to re-evaluate the existing computer-aided design (CAD) methodologies and techniques.

To address the DSM design challenges, one can either increase the lookahead capability of high-level tools or develop new algorithms for solving larger portions of the overall design problem simultaneously. This latter *unification-based approach* is, in our view, more promising. Indeed, the nature of IC design problems and the current state of CAD solutions have reached a point where it is both necessary and possible to combine some steps of the synthesis and physical design processes. The unification-based algorithms are capable of capturing existing interactions among the 'merged' design steps and producing higher-quality implementations by systematically searching a much larger solution space (see [SLP98] and [LSP97]).

The algorithm proposed in this paper integrates two major design steps: fanout optimization and routing tree generation. Each of these two optimization steps has been very effective in reducing the circuit delay, in one case by boosting the transmitted signals via insertion of sized buffers and in the other case by generating suitable wire structures. The goal of this work is to optimally integrate these two steps and thereby provide a uniform framework for optimizing the nets of a placed circuit to achieve faster implementations.

The proposed dynamic programming based algorithm generates and propagates a set of buffered routing tree structures in the form of two dimensional (required time versus input load) solution curves. The resulting wiring structure is guaranteed to inherit the topology of LT-Trees [To90] and the detailed physical implementation of P-Trees [LCLH96]. This algorithm takes a given order for the sinks and, starting from the higher indexed sinks, combines them into groups which are to be driven by buffers. For each group, proper routing structures and buffer locations are examined to generate a set of possible solutions for that subset of ordered sinks. Only the solutions which are not dominated by other solutions are kept. These two steps are repeated in a dynamic programming fashion until the whole set of sinks are combined together. Experimental results reported in this paper demonstrate the effectiveness of this method versus conventional flows that sequentially perform routing tree generation and fanout optimization.

The remainder of the paper is organized as follows. In section 2, background and motivation are given. Section 3 introduces the proposed algorithm. In sections 4 and 5, our experimental results and concluding remarks are presented.

2. BACKGROUND AND MOTIVATION

2.1 Fanout Optimization

Fanout optimization, an operation performed in the logic domain, addresses the problem of distributing a signal to a set of sinks with known loads and required times so as to maximize the required time at the signal driver. Interconnect delay is not incorporated in this operation because the locations of the buffers are not known at this stage. The general fanout optimization problem is NP-hard [To90], however its restriction to some special families of topologies is known to have polynomial complexity.

Among the many existing works on fanout optimization problem, we are interested in the algorithm proposed by [To90]. That work introduces a special class of tree topologies, called LT-Trees, for which the fanout problem is solved with polynomial complexity. The LT-Tree of type-I (in this paper referred to as LT-Tree) is a tree that permits at most one internal node among the immediate children of every internal node in the tree. Touati in [To90] proposed a dynamic programming based algorithm for the fanout optimization problem where the buffer structure is restricted to the LT-Tree topology and sinks with larger required times

*This work was funded in part by SRC under contract no. 98-DJ-606 and by a NSF PECASE award (contract no. MIP-9628999).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
© 1998 ACM 1-58113-008-2/98/0011...$5.00

are placed further from the root of the tree. His algorithm first sorts the sinks in non-decreasing required time order and then starting from the least critical sink, it enumerates all rightmost groupings of the sinks to be driven by a buffer. Finally for each grouping, it enumerates all possible ways of adding either zero or one buffer to drive the rightmost subset of the sinks. Touati gives sufficient conditions for his LT-Tree construction algorithm, LTTREE, to be optimal.

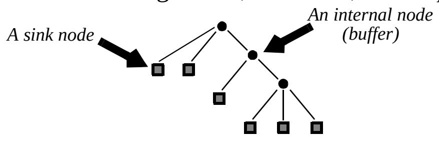

Fig. 1: A simple LT-Tree

Lemma 1: LTTREE works optimally with respect to the signal required time at the root (driver) if all the sinks have equal load capacitances and are sorted in non-decreasing required times [To90].

Lemma 2: LTTREE has $O(n^2)$ polynomial complexity where n is the number of sink nodes [To90].

2.2 Routing Tree Generation

Performance-driven interconnect design, an operation performed in the physical domain, addresses the problem of connecting a source driver to a set of sinks with known loads, required times and positions so as to maximize the required time at the driver. The inherent complexity of this problem has forced researchers to either solve it heuristically or to impose constraints on the structure of the resulting interconnect. For an overview of the existing performance-driven interconnect design technique, interested readers are referred to [CHKM96].

Lillis et al. in [LCLH96] proposed the Permutation-Constrained Routing Tree or P-Tree structure as a solution to the above mentioned problem. Their approach consists of two major phases: Finding a proper ordering for the sinks, and then generating the routing structure based on the calculated ordering. The second phase of the algorithm, called PTREE throughout this paper, is employed in the present paper. Given an ordering of the sink nodes, PTREE finds the optimal embedding of the net into the Hanan grid (the set of points formed by the intersection of horizontal and vertical lines through the terminals of a net [Ha66]) by a dynamic programming approach. In PTREE, the (intermediate) routing solutions are stored in the form of two dimensional, non-dominated solution curves of total area versus required time for every Hanan point.

The worst case complexity of PTREE is rather high, $O(n^5)$, however, the runtime for practical purposes remains within an acceptable range [LCLH96]. Furthermore, by applying some techniques such as controlling the maximum number of Hanan points, the complexity of PTREE is considerably reduced without losing much in terms of quality.

Lemma 3: For a given order on the sinks and with the restriction that the Steiner points lie on the Hanan Grid, PTREE computes the set of all rectilinear Steiner trees with non-dominated required time and total capacitance [LCLH96].

Lemma 4: If the individual capacitive values are polynomially bounded integers or can be mapped to such with sufficient precision, PTREE has $O(n^5)$ pseudo-polynomial complexity where n is the number of sink nodes [GJ79][LCLH96].

Note: Later in section 4, it would be helpful to know that $O(n^2)$ portion of $O(n^5)$ complexity of PTREE is due to the existance of n^2 Hanan points.

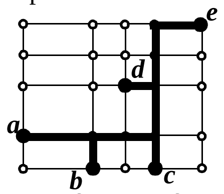

Fig. 2: An output of P-Tree for "*dcba*" order

2.3 Other Works

Okamoto and Cong in [OC96a] proposed a combination of A-Tree routing generation [CLZ93] and van Ginneken's buffer insertion [Gi90] as a solution to the problem of buffered Steiner tree construction. They later extended their work in [OC96b] to include wire sizing as well. Their algorithm takes the placement information of the source and the sinks in addition to the signal required arrival times and then heuristically generates a buffered routing structure such that it maximizes the required time at the source of the net. This technique consists of two phases: bottom up tree construction with non-inferior solution computation and top down buffer insertion. The non-inferior solution which gives the maximum required time at the root is chosen, and then it is traced back through the computations performed during the first phase that led to this solution. During the backtrace, the buffer positions are determined.

During the bottom up phase, the subtrees are combined using a weighted addition function with a user specified parameter to heuristically decide which two subtrees are to be merged. Although this method employs the A-Tree construction algorithm, it cannot guarantee that the resulting structure remains an A-Tree. Furthermore, the fanout optimization algorithm which is based on critical sink isolation is ad-hoc. The overall algorithm has no guarantee of optimality. In contrast, our proposed method produces a buffered rectilinear Steiner tree which is optimal subject to the given order of the sinks, the topology of LT-Trees and the detailed structure of P-Trees.

3. THE FANROUT ALGORITHM

FANROUT, *simultaneous fanout and routing tree optimization algorithm*, is a dynamic programming based algorithm which constructs a buffered routing structure for a given net, based on the available placement, loading, and timing information. The goal is to maximize the required time at the driver of the net.

3.1 Problem Formulation

A given net, $N=(s,S)$, determines the set of sink nodes, $S=\{s_1,s_2,\ldots,s_n\}$, which are to be driven by the driver of the net, called s. In addition to the input net, the following information is required and used by FANROUT:

I. Position of the source $s=(s^x,s^y)$, where s^x and s^y are the horizontal and vertical coordinates of s.
II. Input data for each sink node $s_i=(s_i^x,s_i^y,s_i^l,s_i^r)$ for $1\le i\le n$, where s_i^x and s_i^y are the horizontal and vertical

coordinates, s_i^l is the capacitive load, and s_i^r is the signal required time at node s_i.

III. A library, $L=\{b_1 , b_2 , \ldots , b_m \}$, containing m buffers with different strengths.

IV. A linear ordering of the sinks.

3.2 Two Dimensional Solution Curves

Although the objective is to find an implementation with the maximum required time at s, during every step of FANROUT load versus required time curves are generated and the solutions are compared and evaluated with respect to these two parameters. Comparison of two sub-solutions based on only the required time is an invalid comparison and may result in dropping the optimal solution. This is due to the fact that the loading imposed by a sub-solution on the next level of the LT-Tree may cause a large increase in the overall delay such that the difference between the required times is more than that which was compensated for. Therefore, both the required time and the input load are needed to evaluate the effect of a sub-solution on the overall structure.

Definition 1: Suppose σ_1 and σ_2 are two buffered routing structures for a source and a set of sinks. σ_2 is called inferior to σ_1, if $load(\sigma_1) \leq load(\sigma_2)$ and $reqTime(\sigma_2) \leq reqTime(\sigma_1)$.

3.3 Detailed Approach

FANROUT incorporates LT-Tree and P-Tree construction techniques into a unified framework such that the resulting routing structure is both an LT-Tree, in terms of the overall topology, and a P-Tree, in terms of the detailed physical structure. FANROUT requires an ordering of the sinks and guarantees the optimality of the solution with respect to this ordering only.

In line 1 of Fig. 3, FANROUT loads the subject net, N, which includes a driver, s, and n sinks ordered in some fashion (e.g. based on their placement locations, required times, or a combination thereof). In line 2, it loads the library of the buffers, L, consisting of m buffers with different design parameters, including driving strength, intrinsic delay, and input load. In line 3, $HG(N)$ is loaded with maximum n^2 Hanan nodes which are formed by the intersection of horizontal and vertical lines through the terminals of net N; see Fig. 5.

At every step, z is the index showing that the n-z+1 rightmost sinks (in the ordered list of sinks) are being combined into a group driven by a buffer; see Fig. 4. The LT-Tree topology allows the use of an already processed sub-group of last n-h+1 sinks where h is a number between z and n. This guarantees that in the final solution, each buffer drives directly at most one other buffer.

For every Hanan nodes and every index z, $\Gamma(z,v)$ is a two-dimensional solution curve including all the non-inferior buffered routing structures each connecting sinks s_z through s_n with its root located on v.

In line 4, these solution curves are initialized to the set of all non-inferior buffered paths connecting v to s_n. The code in lines 5 through 16 is for calculating all the buffered routing structures for $\Gamma(z,v)$ using the solutions available in $\Gamma(h,v)$ as described next. Corresponding to group h, there exist n^2 Γ's each for a Hanan node. In line 7, all the Hanan nodes are enumerated by a variable, v, and in line 8, all the solutions in $\Gamma(h, v)$ are retrieved one by one by a variable, γ . So, γ is a routing structure connecting s_h through s_n with its root driven by a buffer located at v. In line 9, PTREE is called on the set of sink nodes (i.e. s_z through s_{h-1}) to be combined with γ ; see Fig. 5. Note that for PTREE, γ acts like a sink node with its corresponding required time and load (where the load is equal to the input capacitance of the buffer driving the routing structure of γ).

```
algorithm FANROUT {
1.  read $N = ( s , S )$ where $s$ is the source and
        $S = \{ s_1 , s_2 , \ldots , s_n \}$ is an ordered list of sinks;
2.  read $L = \{ b_1 , b_2 , \ldots , b_m \}$, the library of buffers;
3.  set $HG( N )$ = all the Hanan grid points of $N$ ;
4.  foreach $v \in HG( N )$ set $\Gamma( n , v )$ = { The set of all
      non-inferior paths from $v$ to $s_n$ };
5.  for $z = n$ to $1$ {
6.    for $h = z$ to $n$
7.      foreach $v \in HG( N )$
8.        foreach $\gamma \in \Gamma( h , v )$ {
9.          set $D$ = PTREE( $HG(N)$ , $\{v, s_z , \ldots , s_{h-1}\}$, $\gamma$ ) ;
10.         foreach $\Delta \in D$ {
11.           set $u$ = Hanan node corresponding to $\Delta$ ;
12.           foreach $\delta \in \Delta$
13.             foreach $b \in L$ {
14.               drive $\delta$ by $b$ and calculate the required
                      time, $r$, at the input of $b$ ;
15.               set $\sigma$ = ( <inputLoad(b) , r> , $\delta$ , $b$ ) ;
16.               add $\sigma$ to $\Gamma( z , u )$ ;
                }
            }
        }
17.   foreach $v \in HG( N )$ prune $\Gamma( z , v )$ ;
    }
18. find ( <l,r>, $\delta$ , $b$ ) $\in \Gamma(1 , v)$ which results in the largest required
      time at the input of the driver, call it bestSolution;
19. retrieve fanroutTree by following the pointers starting
      from the bestSolution ;
20. return fanroutTree ;
}
```

Fig. 3: Pseudo code of FANROUT

PTREE returns a collection of solution curves each corresponding to a distinct Hanan node. The collection of curves is stored in D by PTREE. Then in line 10, these solution curves are selected one by one using a variable, Δ. Recall that each Δ corresponds uniquely to a Hanan node which is referred to as u in line 11. Once a Δ is in hand, its encapsulated routing structures are retrieved one by one by a variable, δ. For all these routing structures, all possible buffers are tried in lines 13 through 16, and for each choice the required time at the input of the buffer is calculated using the specified delay model. In line 15, for every match a solution, σ, is generated (while saving pointers to its sub-solutions, for later use in the top-down traceback phase) which corresponds to a routing structure (i.e., δ) and a buffer (i.e., b). This solution is added to $\Gamma(z, u)$ because the root of σ is located at u . The solution curves $\Gamma(z, v)$ are calculated in this way; however, these curves may contain inferior solutions which are pruned in line 16.

Finally, FANROUT builds the $\Gamma(1, v)$ solution curves (for every v) which contain buffered routing structures connected to all the sink nodes. Then, for every v and for every solution of $\Gamma(1, v)$, the root of the buffered routing structure is connected to the driver and the required time at the input of the driver is calculated in line 18. The structure which results in the largest required time is chosen and is traced down through the stored pointers. The buffered routing structure is retrieved and returned in lines 19 and 20.

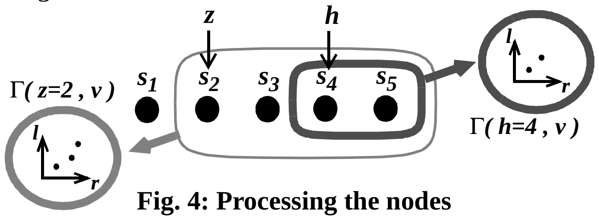

Fig. 4: Processing the nodes

Note that the operations performed in lines 10 through 16, in fact, can be performed internally by a modified PTREE with no increase in the worst case complexity of PTREE. Therefore, in the following complexity analysis we do not take into account the complexity of that part of the pseudo-code.

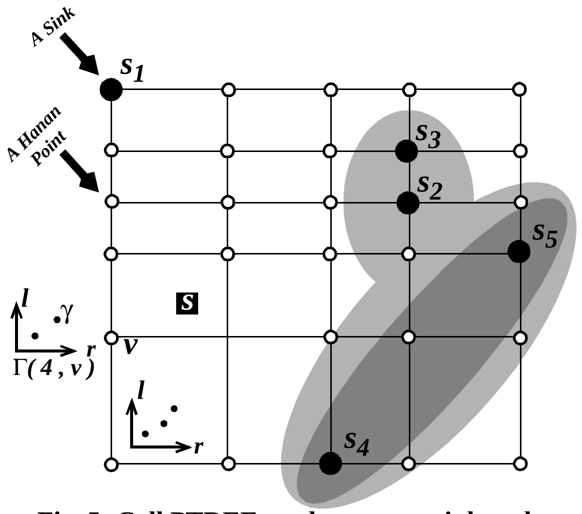

Fig. 5: Call PTREE on the current sink nodes

4. DISCUSSION

4.1 Quality and Complexity of FANROUT

The proposed algorithm is an optimal polynomial algorithm based on a set of assumptions. The following set of lemmas and theorems formally prove these claims.

Theorem 1: The solution space of FANROUT is the product of those of PTREE and LTTREE.

Proof: Any P-Tree structure with inserted buffers such that no buffer immediately drives more than one other buffer can be visited by FANROUT. Also, any LT-Tree such that the output nets of its buffers are implemented using PTREE can be visited by FANROUT.

Lemma 5: For any arbitrary routing with no buffer, $\Re$, which connects the source to the sinks, we have:

I. By decreasing the load of any sink, the capacitance observed at the root of $\Re$ does not increase.

II. By increasing the required time of any sink, the required time at the root of $\Re$ does not decrease.

Proof: For case I, decreasing the load of a sink decreases the amount of charge needed to bring the voltage of $\Re$ to a certain level. For case II, if that particular sink is on the critical path, the statement is trivially true. Otherwise, the required time of the driver is determined by the required time of the other sinks and remains unchanged.

Lemma 6: PTREE is monotone with respect to the load and the required time of the sinks.

Proof: Suppose $\Re$ is a routing structure generated by PTREE. Reducing the capacitance and/or increasing the required time of a sink while preserving $\Re$ results in the decrease of the capacitance and increase of the required time at the root of $\Re$. Therefore, if PTREE is run after changing the load and the required time of the sinks in this way, the resulting structure is non-inferior with respect to $\Re$ and PTREE would store it in the curve (c.f. Lemma 1).

Lemma 7: The use of the pruning operation by FANROUT does not result in the loss of any non-inferior solution.

Proof: Assume that σ_2 is inferior w.r.t. σ_1. By induction, if σ_2 is the whole net and its input is directly connected to the net driver, then the required time does not decrease and the load does not increase by replacing σ_2 with σ_1. If σ_2 is a solution to a sub-problem, its input is driven by another internal node, call it *g*. Due to the monotone behavior of PTREE (c.f. Lemma 6), at *g* the required time and the input load of the implementation including σ_2 is guaranteed to be no better than those of the implementation containing σ_1. A similar argument is then valid for *g* and the rest of the internal nodes down to the leaf nodes.

Theorem 2: FANROUT is an optimal algorithm w.r.t. required time, subject to a set of constraints.

Proof: An examination of the dynamic programming structure of FANROUT shows that if no pruning is performed, all the possible solutions would be considered. Therefore, to prove the optimality of the algorithm it is enough to prove that for an optimal solution, replacing a non-inferior solution with an inferior solution cannot improve the whole implementation; This, however, was proved in Lemma 7.

Lemma 8: The number of solutions in any solution curve is bounded by the number of the buffers in the library, $|L|$.

Proof: The load of any solution is equal to the input capacitance of the driving buffer. However, the number of distinct input capacitances of the buffers is bounded by the total number of the available buffers in the library, $|L|$. For each load value the solution with the maximum required time is stored and the rest will be pruned out.

Theorem 3: FANROUT has $O(n^3)$ memory complexity.

Proof: There are n^2 Hanan points and for each of them n solution curves are stored. Each of the solution curves stores no more than $|L|$ solutions. Therefore, the claim is proved.

Theorem 4: FANROUT has $O(n^9)$ runtime complexity.

Proof: PTREE has $O(n^5)$ worst case runtime complexity (c.f. Lemma 4). Lines 5 and 6 of the pseudo-code, each introduce $O(n)$ complexity and line 7 introduces another $O(n^2)$ complexity. Therefore the overall worst case complexity is $O(n^9)$.

4.2 Reducing the Complexity

Undoubtedly, the worst case complexity of FANROUT is too high for use in many practical cases. However, that complexity can be considerably reduced by applying some simple heuristics. In the following, a couple of heuristics are introduced which are proved to be highly effective with little compromise in terms of the quality of the final results.

I. *Restrict the number of Hanan points:* In the exact version of FANROUT, there are n^2 Hanan points which is a major source of excessive runtime. We may, however, not allow more than *g* Hanan points and change the complexity of line 7 to $O(g)$ and the complexity of PTREE to $O(gn^3)$ (c.f. the note given at the end of sub-section 2.2). Consequently, the worst case complexity of FANROUT is changed to $O(g^2n^5)$.

II. *Bound the maximum number of fanouts driven by a buffer:* We may impose a practical upper bound on the number of fanouts that a buffer drives. Using that value, say *l*, we do not allow FANROUT to connect a buffer to more than *l* fanouts. FANROUT can easily handle this case by changing *n* in line 6 of the pseudo-code to $z+l-1$. In this case the complexity introduced by lines 6 and 9 are changed to $O(l)$ and $O(n^2l^3)$ (c.f. the note given at the end of sub-section 2.2), respectively. Consequently, the worst case complexity of FANROUT is changed to $O(l^4n^5)$.

III. *Fast method:* By applying both of the above technique the complexity of FANROUT is changed to $O(g^2l^4n)$ which results in a linear worst-case complexity when *g* and *l* are assumed to be independent of *n*.

5. EXPERIMENTAL RESULTS

In order to verify the effectiveness of FANROUT, a set of experimental results are reported here. In the presented conventional flows (below), we do not impose any restrictions on the ordering for the sinks. In other words, every fanout optimization and routing tree generation methods are independently free to choose their own appropriate ordering for the sinks (if any needed).

In Table 1, the results are presented for a set of nets taken from a number of benchmarks where the sinks are placed randomly. For these examples, two conventional flows are compared against FANROUT where FANROUT has been used for two different orderings:

I. Ordering with respect to the sink required times, *REQ*.

II. Ordering generated by solving the traveling salesman problem on the set of sinks, *TSP*.

The first conventional flow setup, *conv-I*, uses SIS [SSLM92] for fanout optimization, followed by using PTREE for routing tree generation. For each net, different fanout optimization methods available in SIS are used and for each net only the best result in terms of the required time is reported. The second conventional flow setup, *conv-II*, uses PTREE for routing tree generation followed by using the buffer insertion method introduced in [Gi90]. Note that in Table 1, “total-area”, “req-time” and “w-length” stand for the sum of the area of buffers, the required time at the input of the driver and the total wire length, respectively.

Our next set of experiments (c.f. Table 2) compares the performance of the conventional design flows against our proposed simultaneous algorithm on a number of benchmarks using a CASCADE standard cell library (0.5u HP CMOS process). Gate and wire delays are calculated using a 4-parameter delay equation (similar to that in [LSP97]) and the Elmore delay model [El48], respectively. Also, the fast FANROUT (c.f. sub-section 4.2.) has been run with TSP orderings for the experiments reported in Table 2. These experiments showed that the runtime of the fast FANROUT is in the order of few minutes which is comparable to the runtimes of the conventional flows. Note that the area and delay reported in this table are total chip area and delay after detail routing.

These experiments were run in the SIS environment on an Ultra-2 Sun Sparc workstation (sahand.usc.edu) with 256MB memory.

6. CONCLUSIONS

This paper presents a novel algorithm, FANROUT, which performs simultaneous routing and fanout optimization. It is a dynamic-programming based algorithm which properly uses LT-Tree and P-Tree construction algorithms in order to generate buffer routing structures with maximum signal required time. It computes load versus required time solution curves for every point on the Hanan grid and propagates them while grouping more sink according to the given order. Merge and prune operations are defined on the solution curves to propagate the solution curves through the steps of the algorithm and drop the low quality solutions to maintain the polynomial complexity. FANROUT is an optimal algorithm for maximizing the required time problem for a given order on the sinks. It also inherits all the restrictions that LT-Tree and P-Tree construction algorithms have. FANROUT is a polynomial algorithm as well. This new unified design steps yields high quality circuits in terms of post layout chip area and delay.

7. ACKNOWLEDGEMENTS

We would like to thank Dr. John Lillis of the University of Illinois at Chicago for providing the implementation of the PTREE algorithm and for helpful discussions about the complexity of PTREE.

8. REFERENCES

[CHKM96] J. Cong, L. He, C. Koh, and P. Madden, “Performance optimization of VLSI interconnect layout,” In *Integration, the VLSI Journal 21*, pp. 1-94, 1996.

[CLZ93] J. Cong, K. Leung, and D. Zhou, “Performance-driven interconnect design based on distributed RC delay model,” In *Proceedings of the 30th Design Automation Conference*, pp. 606-611, 1993.

[El48] W. C. Elmore, “The transient response of damped linear network with particular regard to wideband amplifiers,” In *Journal of Applied Physics* 19, pp. 55-63, 1948.

[Gi90] L.P.P.P. van Ginneken, “Buffer placement in distributed RC-tree networks for minimal Elmore delay,“ In *Proceedings of International Symposium on Circuits and Systems*, pp. 865-868, 1990.

[GJ79] [M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman, San Francisco, 1979.

[Ha66] M. Hanan, “On Steiner’s problem with rectilinear distance,” *SIAM Journal of Applied Mathematics*, No. 14, pp. 255-265, 1966.

[LCLH96] J. Lillis, C. K. Cheng, T. Y. Lin, and C. Ho, “New performance driven routing techniques with explicit area/delay tradeoff and simultaneous wire sizing,” In *Proceedings of the 33th Design Automation Conference*, pp. 395-400, 1996.

[LSP97] J. Lou, A. H. Salek, and M. Pedram, ”An exact solution to simultaneous technology mapping and linear placement problem,” In *Proceedings of International Conference on Computer-Aided Design*, pages 671-675, 1997.

[OC96a] T. Okamoto, and J. Cong, “Buffered Steiner tree construction with wire sizing for interconnect layout optimization,” In *Proceedings of*

			Conventional						FANROUT					
			conv-I			conv-II			REQ			TSP		
	nets	# of sinks	req-time	area	w-length	req-time	area	w-length	req-time	area	w-length	req-time	area	w-length
C432	net1	6	35.23	219.12	83.07	21.52	149.93	63.15	21.83	165.66	63.41	17.08	165.66	55.49
	net2	10	33.50	329.12	135.15	29.74	297.66	118.64	28.05	222.20	93.22	24.46	222.20	113.11
C1355	net3	8	36.23	329.12	101.28	25.12	297.66	87.09	32.14	195.47	98.44	29.46	195.47	96.57
	net4	9	34.23	382.58	116.94	30.33	268.18	116.58	28.78	195.47	109.74	26.65	61.82	110.31
C3540	net5	35	38.70	457.49	119.17	38.20	1147.30	152.62	32.16	270.38	122.82	32.01	270.38	132.27
	net6	73	59.44	836.99	535.58	59.78	836.99	549.36	54.75	649.88	549.30	54.69	649.88	583.05
C5315	net7	12	24.94	516.23	68.22	12.21	268.18	42.89	21.83	248.93	71.40	17.23	248.93	70.15
	net8	21	33.10	542.96	195.17	35.59	533.50	254.74	32.61	409.31	206.01	25.32	409.31	200.46
C6288	net9	16	48.33	516.23	144.30	43.75	415.58	168.95	40.38	222.20	157.68	28.96	222.20	160.35
	net10	20	62.49	436.04	146.61	95.96	238.70	175.93	51.67	222.20	136.42	42.86	222.20	145.90
C7552	net11	16	48.57	516.23	179.16	30.28	504.02	211.38	37.83	222.20	182.51	21.98	222.20	171.69
	net12	23	41.68	245.85	185.45	54.88	503.69	261.70	33.00	272.58	157.30	31.62	272.58	189.66
					Average Ratios:		conv-I		0.84	0.63	0.95	0.71	0.61	0.98
							conv-II		1.00	0.70	0.94	0.83	0.66	0.96

Table 1: FANROUT vs. conventional flows for single nets

	Conventional				FANROUT		Ratios			
	conv-I		conv-II		TSP		FANROUT/conv-I		FANROUT/conv-II	
Circuit	Area	Delay	Area	Delay	Area	Delay	Area	Delay	Area	Delay
C17	400.50	0.87	400.50	0.87	416.50	0.90	1.04	1.03	1.04	1.03
C1355	35539.54	10.39	35225.19	10.20	25215.25	7.49	0.71	0.72	0.72	0.73
C1908	51936.70	16.34	48694.77	18.54	43705.90	11.03	0.84	0.68	0.90	0.59
C432	21947.10	11.59	19179.60	13.54	22241.46	11.63	1.01	1.00	1.16	0.86
C499	29203.65	9.27	29208.99	8.99	31201.45	7.17	1.07	0.77	1.07	0.80
C5315	134504.94	19.31	127776.26	20.04	112800.15	10.88	0.84	0.56	0.88	0.54
C880	29786.25	10.53	28626.21	10.20	20811.15	10.01	0.70	0.95	0.73	0.98
alu2	30199.15	14.72	27942.48	17.23	23561.25	10.53	0.78	0.72	0.84	0.61
alu4	50985.15	21.60	46912.67	23.89	51801.75	17.34	1.02	0.80	1.10	0.73
apex6	44626.00	7.12	44514.75	6.67	39516.96	5.27	0.89	0.74	0.89	0.79
cm151a	2042.32	2.88	1753.01	3.21	1560.45	1.83	0.76	0.64	0.89	0.57
dalu	95323.54	23.65	53424.14	26.47	88595.86	23.92	0.93	1.01	1.66	0.90
misex1	4015.55	4.25	3166.56	5.27	3097.44	2.87	0.77	0.68	0.98	0.54
lal	5810.46	3.78	5931.42	4.10	4942.99	2.80	0.85	0.74	0.83	0.68
frg1	6319.74	3.61	6425.50	3.54	5467.56	2.69	0.87	0.75	0.85	0.76
pcle	4775.31	3.51	4644.03	3.53	4161.83	1.89	0.87	0.54	0.90	0.54
rd73	3519.67	3.62	3594.87	3.50	3676.39	3.67	1.04	1.01	1.02	1.05
vg2	5264.19	3.69	5334.03	3.62	5086.35	2.52	0.97	0.68	0.95	0.70
						Average Ratios:	0.89	0.78	0.97	0.75

Table 2: FANROUT vs. conventional flows for a set of benchmarks

International Conference on Computer-Aided Design, pp. 44-49, 1996.

[OC96b] T. Okamoto, and J. Cong, "Interconnect layout optimization by simultaneous Steiner tree construction and buffer insertion," In *Proceedings of the 5'th ACM/SIGDA physical Design Workshop*, pp. 1-6, 1996.

[SLP98] A. H. Salek, J. Lou, and M. Pedram, "A DSM design flow: Putting floorplanning, technology-mapping, and gate-placement together," In *Proceedings of the 35'th Design Automation Conference*, 1998.

[SSLM92] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni-Vincentelli, "SIS: A system for sequential circuit synthesis," *Memorandum No. UCB/ERL M92/41*, Electronics Research Laboratory, College of Engineering, University of California, Berkeley, CA 94720, May 1992.

[To90] H. Touati, "Performance-oriented technology mapping," Ph.D. thesis, *University of California, Berkeley, Technical Report UCB/ERL M90/109*, November 1990.