Routing Tree Construction with Concurrent Performance, Power and Congestion Optimization

Abstract— We present a routing-tree construction algorithm that considers multi-objectives of performance, power and congestion concurrently. Congestion is measured with balanced usage of routing resources among layers. Simultaneous buffer insertion and layer assignment tends to produce routing trees with shorter overall length. Applying the proposed algorithm on a subset of routes on a commercial 64-bit microprocessor yielded 9% less repeater usage and 1.5% shorter overall routing tree length with improved overall performance at the same time, compared to other routing tree construction algorithms.

I. Introduction

The scaling to the *Deep Sub-Micron* (DSM) feature sizes for CMOS VLSI has led to decreased device cost and increased performance. But, starting from the $250nm$ process technology node, the interconnect delay dominates the gate delay [22] due to poor scalability of interconnects. The performance of future VLSI chips will no longer be determined by gate delays rather by interconnect delays. Consequently, such rapid scaling requires performance-driven layout techniques. *Routing-Tree* (RT) synthesis is an increasingly important part of interconnect design.

The typical goal of performance-driven RT construction is to eliminate the *Delay Violation* (DV) to any sink node. The delay violation is defined as the difference between the actual delay and the required delay for a given interconnect. When RT construction is performance-driven, layer assignment and buffer insertion has to be considered concurrently within the RT construction process since they also have a significant impact on delay. Traditionally, layer assignment and buffer insertion are often performed on the finished RT [3, 23, 18, 4, 9, 20], and RT is constructed without the knowledge of buffer insertion and layer assignment. This can yield suboptimal solutions. A performace-driven RT construction algorithm without either buffer insertion or layer assignment or both will try to eliminate the delay violations by trading-off the minimum length RT topology resulting in star-like and longer routing-trees. Attempts have been made recently to incorporate buffer insertion into RT construction [2, 5, 10]

The proposed algorithm incorporates buffer insertion and layer assignment as dual objective in RT construction to achieve delay and power improvement over existing methods. Each step of the RT construction takes advantage of partially constructed, but layer and buffer optimized, RT to construct the next routing segment. While the primary objective of the proposed method is delay reduction, the secondary objectives include minimization of tree length to reduce the routing resource usage; and equal utilization of layer resources to prevent crowding.

Routing Tree construction is NP-Hard [21] and various heuristics are usually employed. Iterative 1-Steiner algorithm [21, 15] starts from a Minimum Spanning Tree (MST) and iteratively modifies the original tree each time introducing a new edge which minimize the total tree length. Minimum Rectalinear Steiner Tree (MRST) minimizes the total tree length which is preferable in terms of routing resource demand. But, they won't necessarily satisfy the timing constraints due to the fact that minimum length routing-tree doesn't necessarily mean the length from source to the critical sink node is minimized.

Jaewon proposed a Linear Programming (LP) based algorithm [13] which has constraints on the source-sink lengths. Even with the minimum critical source-sink path length, the routing tree may not be suitable for VLSI routing since the delay at any sink is not only the function of source-sink length but also the topology of the rest of the tree.

Critical sink routing tree construction algorithms [17, 23, 16] progressively builds the routing tree which, at each iteration, a node is connected (or modified if the starting tree is a sub-optimal tree) so that the maximum delay is minimized (or maximum delay violation or total delay is minimized). Since the delay of a routing tree has to be calculated in order to decide which connection is next in the suboptimal (or partially completed) tree, resistance and the capacitance of the tree edges should be known which is a function of the routing metal layer that tree is being routed. Therefore, layer assignment is generally done without the knowledge of routing tree topology resulting in suboptimal layer assignment.

RTs generated without respect to buffer insertion also yield suboptimal star-like tree topologies. Q-Trees [10] attempts to further optimize the suboptimal buffered trees by *hannan grafting* with a possible buffer replacement. Other algorithms such as P-Tree [24] and C-Tree [25] incorporate multi-objectives of wire sizing, delay/area tradeoff simultanesouly, but assume fixed layer asignment and ignore the impact of buffer insertion on RT topology.

Yildiz's preferred steiner trees [12, 1] assign the edges of the routing tree based on the length. The upper layers become cost effective for long wires while the shorter wires are routed on lower layers due to the additional cost of vias. Although this scheme is desirable, it is not really performance-driven. It is conceivable that a short net can be critical. Furthermore. it can result in overcrowded layers since it doesn't consider congestion. Cho's layer assignment algorithm [7] uses a Network In-

terface Graph (NIG) to model the potential cross-talk between nets. Hence, the nets with high probability of interference are placed on separate layers. NIG doesn't consider timing, hence it can assign critical nets to disadvantaged layers. Saxena [11] suggests assigning the nets that failed timing to the upper routing layer based on layer area quotas during routing process. Most of the existing RT construction algorithms use Ginneken-type dynamic programming based algorithms [18, 9, 8] for repeater insertion during or after RT construction.

In summary, traditional routing flow first builds a routing-tree topology. Layer assignment is performed before or after the routing process. Buffers are usually inserted last for any nets with a delay violation. Some attempts were made [12, 8, 17, 19, 5, 2, 10, 11, 20] to incorporate buffer insertion or layer assignment into the routing process. But no attempt has been made so far to integrate both buffer insertion and layer assignment into the RT construction. Therefore, routing trees are constructed with incomplete information about layer assignment or buffer insertion. The resulting routing trees are often suboptimal. Thus, performance-driven routing tree construction tends to increase the length with star-like tree structures in order to meet the timing requirements. This will greatly increase the routing resource demand and potentially increase use of repeaters. Moreover, the resulting routing-tree topology may also uneven layer resource usage due to failed nets being reassigned to other layers during the rip-up and reroute process.

In this paper, we present a new routing tree construction algorithm that concurrently performs layer assignment and buffer insertion during the process of routing tree construction. The proposed algorithm generates more compact routing trees with less buffers and less congestion at different layers while producing better delays. Delay calculation during RT construction uses AWE-based methods [14] for accuracy.

The remaining part of this paper is organized as follows: Section II describes the proposed routing tree construction algorithm with simultaneous layer assignment and buffer insertion. Experimental results using a 64-bit microprocessor core is given in Section III. Finally summary and concluding remarks are presented in Section IV.

II. Routing-Tree Construction with Simultaneous Layer Assignment and Buffer Insertion

The algorithm of RT construction with Layer Assignment and Buffer Insertion (LABI) is shown in Figure 1. For a given netlist, the algorithm starts from the source node of each net and iterates through every sink node until the entire tree is constructed for every net in the netlist. At each iteration, *BuildPartialTree* chooses the most timing critical sink node from the unconnected sink nodes. If a net is a non critical net, i.e. there is no delay violation, *BuildPartialTree* chooses the sink node which minimize the total tree length. The most critical sink node is decided by tentatively connecting every sink node to the partial tree, and by calculating the maximum delay violation of the tree. *BuildPartialTree* joins the critical sink node

```
1  LABI(S)
2  begin
3    S = All Nets
4    InitialLayerAssignment(S)
5    n_net = unconnectedSinks(net), net ∈ S
6    while (n_net ! = φ, ∀ net ∈ S)
7          #Start BuildPartialTree
8          foreach net ∈ S
9            m_net = partialTree(net)
10           n_net = unconnectedSinks(net)
11           if (net is critical)
12             s = criticalSink(m_net, n_net)
13           else
14               s = minLengthSink(m_net, n_net)
15           m_net = m_net + s;
16           n_net = n_net − s;
17           GrowTree(m_net, s)
18         #End BuildPartialTree
19         do
20            AdjustLayerAssignment(S)
21           foreach net ∈ S
22             m_net = BldpartialTree(net)
23             InsertBuffer(m_net)
24         while (Buffer inserted for any net in S)
25 foreach net ∈ S
26   foreach buf ∈ net
27     RemoveBuffer(net, buf)
28     if (DV_net > 0)
29       ReinsertBuffer(net, buf)
30 end
```

Fig. 1. Simultaneous buffered routing tree construction with layer assignment

to the existing tree such that the maximum delay violation of the tree is minimized or the total length of the tree is minimized if there is no delay violation. Althought any progressive RT construction can be adapted here, this scheme is similar to the *Critical Sink Routing Tree* (CSRT) approach [16] with *HBest* variant and without *global Slack Removal* (GSR) post processing. The major difference lies in the order of unconnected sinks to be chosen to connext to the tree: We choose the most critical sink that minimize the delay violation whereas GSR chooses sink which minimizes the tree delay is chosen. Our preliminary experiments indicated slight improvement with our scheme. Since the timing of each net has been changed at each iteration by growing the tree, layer assignment of the entire netlist is readjusted and buffers are inserted. When the buffer insertion and layer assignment are applied to the partial tree, It will update the timing of the partially constructed tree allowing the next sink node to connect to a better location. In other words, by inserting buffers or improving the delay of the tree with layer assignment, sink nodes can potentially connect to the lower edges in the tree topology (or to a closer edge to the sink), effectively reducing the total tree length. The post processing phase (lines 25-29) at the end will be discussed later.

A simple example to compare the RT construction process is given in Figure 2 where *rat* is *required arrival time* and *dv* is *delay violation*. The traditional methods, where RT construction, layer assignment and buffer insertion are performed separately, produce the star-like topology as shown in the Figure

2 a). The proposed method, the buffers inserted into the partial tree in Figure 2 b), enables the subsequent connections to better locations yielding more compact and higher performance tree topology.

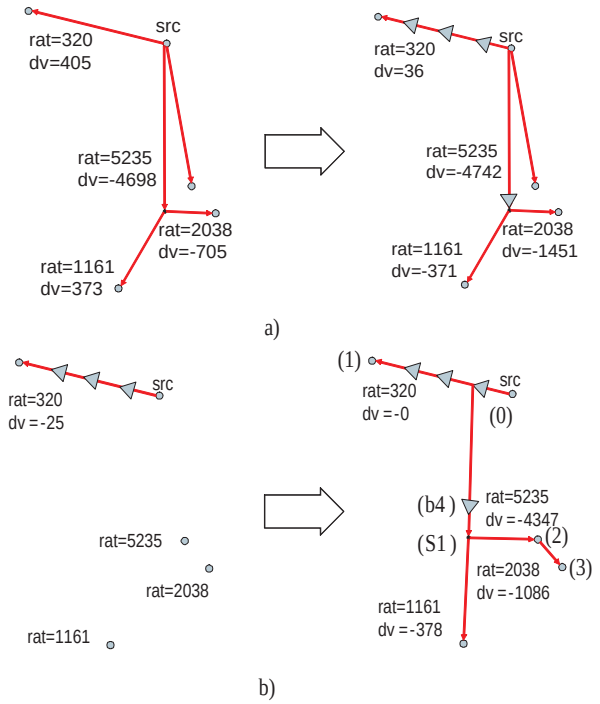

Fig. 2. a)RT construction, layer assignment and buffer insertion b)Concurrent RT construction, layer assignment and buffer insertion

It is clear from this example that simultaneous consideration of different objectives during RT construction can produce better routes. No single task, i.e building tree, layer assignment, and buffer insertion, takes in its entirely without considering its impact on other tasks and the overall routing quality. The following subsections give more details about different aspects of the RT construction process.

A. Routing-Tree Construction

The proposed algorithm for *BuildPartialTree* shown in Figure 3 uses a greedy approach to progressively build the routing tree with minimum delay violation. Initially, the tree includes only the source node. At each iteration, all the unconnected sink nodes are tentatively connected to the existing tree and the most critical sink node is chosen and permanently connected to the existing tree. In the proposed algorithm, the preference is given to the most timing critical sink node to be connected first. In the existing RT construction algorithms, preference has been given to the least timing critical sink [16, 23]. Although there is no clear preference based on theoretical advantages given in the literature, our own experiments indicate a slight advantage of giving preference to the most timing critical sink. This can be seen from the example in Figure 2 b): The buffer insertion (buffer 4) isolates the most critical path (path 0-1) in the

```
1  BuildPartialTree(T)
2   #one iteration
3  bestDV = −INF
4  bestLen = INF
5  foreach unconnected node n ∈ T
6     bestDVPin = INF
7     bestLenPin = INF
8     foreach edge e ∈ T
9        JoinNodeAtEdge(T, n, e);
10       awe(T);
11       if (DV(T) < 0  &  Len(T) < bestLenPin)
12          bestLenPin = Len(T)
13          bestEdge = e
14       elsif (DV(T) < bestDV)
15             bestDVPin = DV(T)
16             bestEdge = e
17       DisjoinNode(T, n)
18    if (DV(T) < 0  &  bestLenPin < bestLen)
19       bestLen = bestLenPin
20       bestNode = n
21    elsif (besDVPin > bestDV)
22          bestDV = bestDVPin
23          bestNode = n
24 JoinNodeAtEdge(T, bestNode, bestEdge)
```

Fig. 3. Performance driven progressive routing tree construction

early stages allowing less critical sinks (sink 2,3) to connect a location (s1) which is not on the critical path. The progressiveness of the proposed RT construction allows concurrent buffer insertion and layer assignment as an integral part of the RT construction algorithm. The proposed algorithm falls back to minimum length tree construction if there is no delay violation.

For a partially constructed tree with i pins, the maximum number of edges is $2i-3$ and the remaining number of pins is $n-i$ where n is the total number of pins. The possible permutations of tree to connect the rest of the pins can be expressed as:

$$\sum_{i=1}^{n}(2i-3)(n-i) \tag{1}$$

Hence, the complexity of the proposed RT construction algorithms is $O(n^3)$ due to the inner search loop for the most critical pin.

B. Buffer Insertion and Removal

The buffer insertion problem is NP-hard [9, 18]. Equal distance buffer insertion generally yields good *delay reduction-complexity* trade-off [23]. This is due to the fact that if the distance between buffers d_i is equal to one other, $\sum d_i^2$ is minimized provided that $\sum d_i$ is constant [6]. The proposed algorithm inserts a buffer which maximizes the delay violation reduction by evaluating delay reduction with respect to possible buffer locations. The algorithm evaluates buffer locations at Steiner points as well as on routing segments.

The overall buffer insertion algorithm, shown in Figures 4 and 1, consist of two phases. During the first phase, *InsertBuffer*, buffers are inserted for critical nets in the netlist. The second phase, where excessive buffers are removed as shown

```
1  InsertBuffer(T)
2  bestDV = INF
3  foreach edge e ∈ T
4    b = InsertBufferAtEdge(T, e)
5    awe(T)
6    if (DVImp(T) > 10ps  &  DV(T) < bestDV)
7      bestEdge = e
8      bestDV = DV(T)
9    RemoveBuffer(T, b)
10 InsertBufferAtEdge(T, bestEdge)
```

Fig. 4. Buffer Insertion

in Figure 1 (lines 25-29), re-optimizes the RTs by further trading off performance against power. At each iteration during the first phase, a buffer is tentatively inserted into each candidate buffer location. The buffer location with the best delay violation improvement is chosen. The iteration loop stops when the amount of delay improvement is below a preset threshold for all possible buffer locations, or the timing constraints for the net are met. When the first phase is completed, the second phase starts by re-visiting every net once to perform further performane power and delay trade-off. This is a desirable step due to the fact that inserted buffers on a partially built tree may be suboptimal after the rest of the tree is completed. The complexity of the overall buffer insertion algorithm is $O(n)$.

C. Layer Assignment

Layer assignment can be used to improve routing quality in three different ways:

- Delay for the same length wires is better on the upper layers. Hence, critical nets can be assigned to the upper layers to improve timing.
- Coupling is usually less between layers. Therefore, crosstalk can be reduced by placing high-interference nets into separate layers.
- If a routing area on a particular layer is highly congested, some nets can be assigned to a different layer to reduce the congestion.

The proposed algorithm performs an initial layer assignment for all the nets before the routing tree construction starts. Although the accurate timing, congestion and cross-talk information are not available prior to routing, an estimated metric can be used to assign a net to a layer to improve the routing. Our algorithm uses estimated layer usage to calculate the utilization of the layer. Layer usage is defined as:

$$U_l = \sum_{i \in B} b_i^l / pitch^l + \sum_{j \in N+S} d_j^l \qquad (2)$$

The layer usage incurred from blockages is calculated as the sum of the track lengths covered by the blockage. b_i^l is the area of the blockage i at layer l. $pitch^l$ is the distance between routing tracks at layer l. Layer usage incurred from existing routing N and current routing S is simply calculated by adding the horizontal (or vertical) distance d_j^l of the routing tree edges of each net. The proposed assignment algorithm relies on sort

```
1  LayerAssginment(S)
2  #S=net list
3  #initialize layer usages
4  U_l = 0, 1 ∀ layers
5  U_l = U_l + addBlockages()
6  U_l = U_l + addExistingRouting()
7  do
8     x = S.end()
9     y = x − 1
10    do
11       x = x − 1
12       y = y − 1
13       if x and y are on the same layer
14         if DV_x < DV_y
15           swap(x, y)
16       elsif x and y are on neighboring layers
17            if U_{l_x} > 1.1 * U_{l_y}
18              l_x = l_y
19            elsif U_{l_x} < 0.9 * U_{l_y}
20                 l_x = l_y
21            elsif max(DV_{x^{l_y}}, DV_{y^{l_x}}) <
22                 max(DV_x, DV_y)
23                 swap(x, y)
24                 updateLayerUsage(U_{l_x})
25                 updateLayerUsage(U_{l_y})
26    while y ≠ S.begin()
27 while swapped
```

Fig. 5. Performance-Driven Layer Assignment

list created based on routing capacity of each metal layers as shown in 6. Each metal pair occupies a segment of hte sort list. Teh lower metal layers sit towards the bottom of the sort list. Inititally, all nets are randomly inserted into the list. the nets in the list are sorted based their delay violations (DV). If the layer assignment of both net is the same, the net with the higher delay violation bubbles-up, If the nets are in different layers, layer usages are first compared:

- If the upper layer's usage is bigger than lower layer's usage, both net are assigned to the lower layer.
- If the layer usage is similar, nets are swapped tentatively to compare the maximum delay violation. The assignment that yields the best delay violation is chosen.

The process is repeated until there is no legal swap left in the list. Fig 5 gives the layer assignment algorithm. The resulting layer assignment moves critical nets to the upper metal layers. However, this is counter balanced by the requirement that the layer usage distribution be as homogeneously as possible among different layers to obtain less coupling and less congestion when the routing is actually performed. It is worth noting that nets in the list are swapped throughout the routing process in concurrent fashion with repeater insertion and RT construction.

Althgouht the worst case complexity of the sort algorithm is $O(n^2)$, sort is usually completed around $O(n)$ except during the first iteration where nets are randomly inserted into the

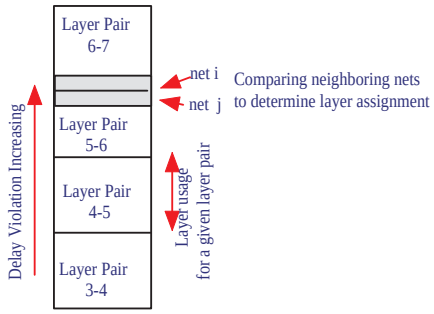

Fig. 6. DV sort list for layer assignment

list since delay (hence criticality) is unknown initially. In the subsequent iterations, list is mostly sorted and swap operation doesn't occur frequently.

III. Experimental Results

The proposed algorithm was applied to construct routing trees for a subset of the glonal nets in a 64-bit microprocessor core. The subset has a total of 13204 unconnected nets. In order to understand the behavior of the proposed RT construction algorithm, different types of multiple-pin nets, i.e. 2-pin nets and 3-pin nets, were extracted from the unrouted nets to form different categories for comparison. There are a total of 9938 2-pin nets and 1339 3-pin nets. Since there are not enough nets with more than 3 pins in the unrouted set. Therefore 1000 randomly generated 4 to 6-pin nets were used for comparison. For the nets in the microprocessor core, the timing constraints were extracted from a static timing database. For randomly generated nets, the timing constraints were generated based on timing of the corresponding minimum length Steiner tree plus a random perturbation. The routing uses five upper metal layers.

RT constructions were performed on each set of routes. The unrouted nets of the microprocessor core as a whole were also considered as a test case. Since the CSRT method [16] is a typcal sequental algorithm and is widely used for routing tree construction, we compare the results from the proposed concurrent algorithm to those from CSRT. Because CSRT doesn't include buffer insertion and layer assignment, for the sake of fairness, the proposed buffer insertion algorithm is applied to the nets with delay violation at the end. An initial layer assignment is performed based on estimation of timing criticality of the nets. This is a typical flow in a highly hierarchical custom design. The initial layer assignment is often performed manually.

The results were compared based on the following metrics:

- $\#B$: number of buffers inserted.
- $\#DV$: number of nets with delay violation.
- $maxDV$: maximum delay violation.
- L : total length of the routing-trees.
- CPU : CPU time.

The results in Table I show that the number of delay violations are consistently better in each test case for the proposed algorithm. The number of buffers required for achieving better delay violations are also consistently less for the proposed algorithm, ranging from 9% less buffers for μP core to around 29% for 6-pin nets. The proposed algorithm also achieved around 1.5% shorter overall routing tree length. For 2-pin nets, the proposed algorithm is reduced to the traditional method since there is only one possible connection for 2-pins nets. The propsed algorithm performs better overall as the pin count increases. This is due to fact that critical pins are optimized early in the routing process and less critical pins are connected to better locations on the existing routing tree later in the routing process. The observation has been illustrated earlier in the paper in Figure 2.

TABLE I
Sequential and simultaneous routing tree construction results

		$\#B$	$\#DV$	$maxDV$ (ps)	L (m)	CPU (s)
9938/2-pins	CSRT	2339	277	205	22.6	27
	Our Alg.	2339	277	205	22.6	40
1339/3-pins	CSRT	1129	184	208	6.5	1.9
	Our Alg.	979	181	201	6.4	2.8
1000/4-pins	CSRT	685	63	348	16.5	2.2
	Our Alg.	490	53	214	16.5	3.8
1000/5-pins	CSRT	783	53	267	19.7	3.2
	Our Alg.	699	40	244	19.5	4.1
1000/6-pins	CSRT	1041	56	470	23.1	7.3
	Our Alg.	735	41	309	22.5	8.2
13204 nets	CSRT	3690	535	221	32.1	69
μP core	Our Alg.	3335	533	201	31.7	159

Table II shows how the wires are distributed among layers for the μP core test case. The proposed algorithm yields more homegenous layer utilization. This can potentially yield better congestion optimization for the global routing. Since the actual routing has not been performed at this stage, wire lengths are used as a congestion metric instead of overflows on routing tiles.

IV. Summary and Conclusions

A new multi-objective based routing-tree construction technique considering performance, power and congestion has been presented. The proposed algorithm incorporates buffer

TABLE II
WIRE DISTRIBUTION

	$M3$	$M4$	$M5$	$M6$	$M7$
CSRT	12%	28%	23%	27%	10%
Our Alg.	17%	22%	24%	22%	15%

insertion and layer assignment simultaneously into the RT construction process. The experiments on 2-pin to 6-pin nets shows that the proposed algorithm consistently produces shorter tree length, less number of buffers and less number of delay violations. The higher fanout of a net, the better improvement the proposed algorithm can provide.

V. ACKNOWLEDMENT

The research presented in this paper was funded partially by HP through an HP Graduate Fellowship. The authors would like to thank HP for their generous support throughout this research

REFERENCES

[1] Ameya R. Agnihotri, Patrick H. Madden, "Congestion reduction in traditional and new routing architectures," *Proceedings of the 13th ACM Great Lakes Symposium on VLSI*, pp. 211 - 214, April 2003.

[2] Hai Zhou; D.F. Wong; I-Min Liu; A. Aziz, "Simultaneous routing and buffer insertion with restrictions on buffer locations," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, pp. 819 - 824, July 2000.

[3] S. Muddu; E. Sarto; M. Hofmann; A. Bashteen, "Repeater and interconnect strategies for high-performance physical designs," *Integrated Circuit Design, 1998. Proceedings. XI Brazilian Symposium on*, pp. 226 - 231 , Oct 1998.

[4] Jiang Hu; C.J. Alpert; S.T. Quay; G. Gandham, "Buffer insertion with adaptive blockage avoidance," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, pp. 492 - 498, April. 2003.

[5] Li-Da Huang; Minghorng Lai; D.F. Wong; Youxin Gao, "Maze routing with buffer insertion under transition time constraints," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, pp. 91 - 95, Jan 2003.

[6] P. Saxena; B. Halpin, "Modeling Repeaters Explicity Within Analytical Placement," *Design Automation Conference, 2003. Proceedings*, pp. 699 - 704, June 2004.

[7] C.D. Cho; S. Raje; M. Sarrafzadeh; M. Sriram; S.M. Kang, "Crosstalk-minimum layer assignment," *Custom Integrated Circuits Conference, 1993., Proceedings of the IEEE 1993*, pp. 29.7.1 - 29.7.4 , May 1993.

[8] M. Hrkic; J. Lillis, "Buffer tree synthesis with consideration of temporal locality, sink polarity requirements, solution cost, congestion, and blockages," *IEEE Transactions on CAD*, pp. 481 - 491, April 2003.

[9] J. Lillis; Chung-Kuan Cheng; T.-T.Y. Lin, "Optimal wire sizing and buffer insertion for low power and a generalized delay model," *Solid-State Circuits, IEEE Journal of*, pp. 437 - 447, March 1996.

[10] A.B. Kahng; Bao Liu, "Q-Tree: a new iterative improvement approach for buffered interconnect optimization," *VLSI, 2003. Proceedings. IEEE Computer Society Annual Symposium on*, pp. 183 - 188, Feb. 2003.

[11] P. Saxena; C.L. Liu, "Optimization of the maximum delay of global interconnects during layer assignment," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, pp. 503 - 515, April 2001.

[12] M.C. Yildiz; P.H. Madden, "Preferred Direction Steiner trees," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, pp. 1368 - 1372, Nov. 2002.

[13] Jaewon Oh; Iksoo Pyo; M. Pedram, "Constructing lower and upper bounded delay routing trees using linear programming," *Design Automation Conference Proceedings 1996, 33rd*, pp. 401 - 404, June 1996.

[14] N. Gopal; D.P. Neikirk; L.T. Pillage, "Evaluating RC-interconnect using moment-matching approximations," *Computer-Aided Design, 1991. ICCAD-91. Digest of Technical Papers, 1991 IEEE International Conference on*, pp. 74 - 77, Nov. 1991.

[15] A.B. Kahng; G. Robins, "A new class of iterative Steiner tree heuristics with good performance," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, pp. 893 - 902, July 1992.

[16] K.D. Boese; A.B. Kahng; B.A. McCoy; G. Robins, "Near-optimal critical sink routing tree constructions," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, pp. 1417 - 1436, Dec. 1995.

[17] Jiang Hu; S.S Sapatnekar, "A timing-constrained simultaneous global routing algorithm," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, pp. 1025 - 1036, Sept. 2002.

[18] Van Ginneken, L.P.P.P., "Buffer placement in distributed RC-tree networks for minimal Elmore delay," *Circuits and Systems, 1990., IEEE International Symposium on*, pp. 865 - 868, May 1990.

[19] T. Deguchi; T. Koide; S. Wakabayashi, "Timing-driven hierarchical global routing with wire-sizing and buffer-insertion for VLSI with multi-routing-layer," *Design Automation Conference, 2000. Proceedings of the ASP-DAC 2000*, pp. 99 - 104, Jan. 2000.

[20] C.C.N. Chu; D.F. Wong, "A quadratic programming approach to simultaneous buffer insertion/sizing and wire sizing," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, pp. 787 - 798, June 1999.

[21] J. Griffith; G. Robins; J.S. Salowe; Tongtong Zhang, "Closing the gap: near-optimal Steiner trees in polynomial time," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, pp. 1351 - 1365, Nov. 1994.

[22] www.src.com, "International Technology Roadmap for Semiconductors," *Semiconductor Research Corporation*, 2003.

[23] Jiang Hu; S.S. Sapatnekar, "Algorithms for non-Hanan-based optimization for VLSI interconnect under a higher-order AWE model," *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, pp. 446 -458, April 2000.

[24] J.Lillis; C.K.Cheng; T-T.Y.Lin; C-Y.Ho, "New Performance Driven Routing Techniques With Explict Area/Delay Tradeoff and Simultaneous Wire Sizing," *33th Design Automation Conference Proceedings*, pp. 395 -400, June 1996.

[25] D.Wang; E.S.Kuh, "A new General Connectivity Model and Its Applications to Timing-Driven Steiner Tree Routing," *Electronics, Circuits and Systems, 1998 IEEE International Conference on*, pp. 71 -74, Sept 1998.