Crosslink Insertion for Variation-Driven Clock Network Construction

Fuqiang Qian, Haitong Tian, Evangeline Young
Department of Computer Science and Engineering
The Chinese University of Hong Kong
{fqqian, httian, fyyoung}@cse.cuhk.edu.hk

ABSTRACT

Link based non-tree clock network is an effective and economic way to reduce clock skew caused by variations. However, it is still an open topic where links should be inserted in order to achieve largest skew reduction with smaller extra resources. We propose a new method using linear program to solve this problem in this paper. In our approach, clock skew in a non-tree clock network is computed using the delay model in [13] and the information is used to select the node pairs for link insertion. Tradeoff between crosslink length and skew reduction effect is explored. Based on the analysis, we propose a new algorithm to insert crosslinks into a clock network. We compare our work with the method in [1] and a recent work [4] which inserts links between internal nodes of a tree. Experiments show that our method can reduce skew under variations effectively.

Categories and Subject Descriptors

B.7.2 [**INTEGRATED CIRCUITS**]: Design Aids---Placement and routing

General Terms

Algorithm, Design

Keywords

Clock, Skew, Non-tree, Crosslink

1. INTRODUCTION

Clock skew limits the performance of a synchronous digital system. On the other hand, the effect of process variation is becoming more significant. Clock skew caused by variations is now one of the most critical problems that the design of large and high performance system is facing today.

Non-tree clock network is a promising way to address the skew variation problem. Clock mesh, because of its inherent redundancy, is more tolerant to process variation and is able to provide lower skew variability compared with traditional clock tree. However, clock mesh has the disadvantage of resulting in much more power dissipation. A clock distribution network that combines the advantages of tree and mesh is described in [5]. The design is used on several microprocessor chips, achieving very low skew. For analysis and optimization, Venkataraman *et al.* [6] managed to reduce a clock mesh by retaining only the critical edges. Their strategy offers designers a possibility of trade-off between power and tolerance to process variation. In paper [7], a sliding window based scheme is used to analyze the latency in a clock mesh. A comprehensive and automated framework for planning, synthesizing and optimization of clock mesh networks is proposed in [8].

Compared to the mesh structure, clock network constructed by inserting crosslinks will consume much less power. Rajaram *et al.* [1] propose a framework to construct such non-tree network and present an analysis on the effect of link insertion on skew variability. Lam *et al.* [9] present a statistical based non-tree clock network construction technique. They use statistical timing analysis to obtain the distribution of skew in a clock tree network. The works in [10, 11] explore the challenges in buffered clock tree link insertion and construct a buffered clock network with crosslinks. In [12], wire sizing is performed to improve skew variability in a non-tree topology with links which are generated using the minimum weight matching-based method in [1]. Recently, a link insertion scheme that inserts crosslinks at higher level internal nodes instead of sink nodes in a clock tree is proposed [4]. Their work reduces skew variability and total crosslink length.

Although there are previous works on non-tree clock network construction with crosslinks, some important questions are still unanswered. Most existing works on link insertion attempt to reduce skew variability while use the minimum wirelength. However, the tradeoff between the length of a link and its ability to reduce clock skew is not analytically studied. In this work, efforts are made towards solving these problems. We use the load redistribution and tree decomposition technique [13] to obtain the delay and skew values in a non-tree clock network. We then formulate the crosslink insertion problem as a sequence of linear programs, with an objective to find a pair of nodes to insert a crosslink such that the skew variability can be reduced the most while the wirelength increase due to the link insertion is constrained. By applying this technique recursively, we can add a user-defined number of crosslinks and the clock skew will be reduced progressively. Simulation results show that our method can lead to significant skew reduction under variations. The power consumption is also reduced comparing with previous works. The major contributions of this paper can be summarized as follows:

• Our work computes signal delay and clock skew in a non-tree clock network analytically when inserting links. The link insertion process thus produces more effective result.

• We formulate the problem of finding a node pair for link insertion in a non-tree clock network as a sequence of linear optimization problems, with an objective function to consider the tradeoff between clock skew reduction and link capacitance.

• We devise a method to speed up the linear program so that even for a clock network with several thousand sinks, the problem can be solved in a reasonable amount of time.

The paper is organized as following. Techniques for computing signal delay and clock skew in non-tree clock networks will be discussed in Section 2. In Section 3, our method to construct a non-tree clock network with crosslinks will be presented. The experimental results will be reported in Section 4. Finally, concluding remarks will be made in Section 5.

2. SIGNAL DELAY AND CLCOK SKEW IN NON-TREE CLOCK NETWORKS

In this section, the delay calculation technique in general *RC* network will be reviewed. This method can be used to evaluate node delays and clock skew in a non-tree network with crosslinks. Some analysis of the load redistribution effect of a link will be provided.

Calculation of the delays in a non-tree network is non-trivial, because there are loops between the nodes, which makes possible that one node is driving and loading another node at the same time. The relationships between nodes are not explicit in a non-tree clock network. Several techniques have been suggested to model the delay in general *RC* network [3, 13, 14].

Based on the Elmore delay model, a technique called tree decomposition [13] is proposed to calculate the delays of all the nodes in a general *RC* network with resistance loops. Suppose that N is an arbitrary node in a general *RC* network and N has k neighboring nodes, denoted by M_i where $i = 1, ..., k$. Node N is connected to M_i through edge E_i with a resistance of value R_i. Denote the load capacitance of N as C. The idea is to partition and distribute C into N's k neighboring edges. Let C_i be the equivalent load distributed to edge E_i, note that C_i can be negative. According to the calculation of the Elmore delay, if the delay of node M_i is T_i, the delay of node N will be $T = T_i + R_iC_i$. Then there is a set of constraints as follows.

$$\Sigma_{i=1,\dots,k} C_i = C \tag{1}$$

$$T = T_i + R_iC_i \qquad i = 1, ..., k \tag{2}$$

Applying the above idea, a general *RC* network can be decomposed into a tree. Since the delays in a tree can be determined efficiently, we are able to calculate the delays and clock skews in the original *RC* network.

Consider the effect of inserting a link between two nodes N_i and N_j with delays D_i and D_j respectively in a tree as shown in Fig.1(a). Assume that the load capacitances of node N_i and N_j are C_i and C_j respectively before link insertion. After a link is inserted between node N_i and N_j, node N_j becomes one of the neighboring nodes of N_i. The loading capacitance of node N_i will be redistributed as in Fig.2(b). In this example, the load capacitance of node N_i is redistributed. Note that we can also split N_j instead of N_i. Suppose $C_{i,1}$ is the capacitance remaining at node N_i and $C_{i,2}$ is the load redistributed from node N_i to node N_j through the inserted link. Let $D_i^{'}$ and $D_j^{'}$ be the new delays at node N_i and N_j respectively, which can be calculated after this decomposing step.

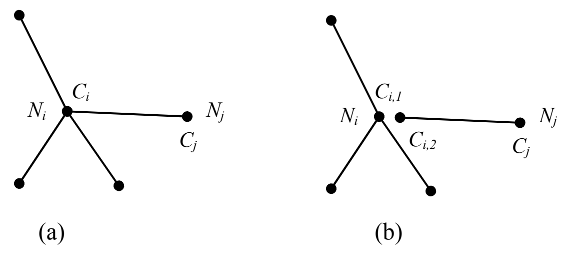

Fig.1. Load Redistribution through a Link

According the set of constraints (1) and (2), we have the following equality constraints for this particular example:

$$C_{i,1} + C_{i,2} = C_i \tag{3}$$

$$D_i^{'} = D_j^{'} + D_{link} \tag{4}$$

The delay difference between node N_i and node N_j becomes $D_{link} = R_{link}C_{i,2}$. Note that if $C_{i,2}$ is positive, it indicates that node N_j is actually driving node N_i after this link insertion step and node N_j load part of node N_i's capacitance, which is $C_{i,2}$. If $C_{i,2}$ is negative, it indicates that node N_i is driving node N_j instead. This load redistribution is also applicable for node pairs with non-zero skew.

The Elmore delay of node N_i in a tree T is

$$\sum_k R_{i,k} C_k \tag{5}$$

where $k \in$ all nodes in T, $R_{i,k}$ is the shared path resistance between node N_i and node N_k and C_k is the capacitance of node N_k. For a given tree, the resistances of edges and capacitances of the nodes are known, so the delays at nodes N_i and N_j can be written as follows with constants k, where $i = 1, 2, 3, 4, 5, 6$.

$$D_i = k_1C_i + k_2C_j + k_3 \tag{6}$$

$$D_j = k_4C_i + k_5C_j + k_6 \tag{7}$$

The skew between node N_i and N_j before link insertion is

$$q_{ij} = D_i - D_j \tag{8}$$

After a link is inserted between N_i and N_j, the delays become

$$D_i^{'} = k_1(C_i - C_{i,2}) + k_2(C_j + C_{i,2}) + k_3 \tag{9}$$

$$D_j^{'} = k_4(C_i - C_{i,2}) + k_5(C_j + C_{i,2}) + k_6 \tag{10}$$

The skew becomes $q_{ij}^{'} = D_i^{'} - D_j^{'}$, given by

$$q_{ij}^{'} = q_{ij} - (k_1 - k_4)C_{i,2} + (k_2 - k_5)C_{i,2} \tag{11}$$

which equals $R_{link}C_{i,2}$, so we have

$$C_{i,2} = q_{ij}/(k_1 - k_2 - k_4 + k_5 + R_{link}) \tag{12}$$

The skew after link insertion is $q_{ij}^{'} = R_{link}C_{i,2} = R_{link}\, q_{ij}/(k_1 - k_2 - k_4 + k_5 + R_{link})$, the skew is thus scaled by $R_{link}/(k_1 - k_2 - k_4 + k_5 + R_{link})$. Therefore, the skew reduction effect of a link is affected by the resistance of the link R_{link}. If a link resistance is large, its skew reduction effect will be small. Although the above analysis is based on a structure that is a tree, the same argument can be applied even after several links are inserted since a non-tree network can be represented by a tree structure by applying the above load redistribution method.

3. LINK INSERTION FOR NON-TREE CLOCK NETWORK

Based on the techniques discussed in section 2, we are able to calculate signal delays in a clock network. In our approach, the sink load capacitances are modeled as variables because they are affected by various variations. Each sink node's delay will change according to the values of the sink capacitances. We use the worst case skew value as a metric to evaluate the skew. A worst case skew is defined as the maximum skew that might appear in the clock network with variations under consideration.

Adding links between the node pairs which are most susceptible to have the worst case clock skew will be beneficial. Therefore, one important step is to identify the node pairs which will have large skew under variations. A shared path length l_{ij} is the length of the path shared between the path from the root s to sink node S_i and the path from s to S_j. It is obvious that if two nodes share a long common path from the source in a tree topology, the delay difference will be small. If the shared path length l_{ij} is small or even zero, the delay difference between S_i and S_j can be high. Those node pairs with small shared path length are topologically far away. In a clock tree, adding a link to those topologically far away node pairs will be beneficial because they are likely to have large clock skew. For example, in Fig. 2, adding a link between node *b* and node *c* will generally be better than adding a link between node *b* and node *d*, because node pair (*b, d*) is less likely to have large skew.

When the clock network is no longer a tree structure after some crosslinks are inserted, the relationship between nodes will not be explicit. It is hard to tell if a node pair is topologically far away or not. In Fig. 2, if we consider node pair (*a, b*) and node pair (*d, c*), node pair (*a, b*) are topologically closer if no links are inserted, since the lowest common ancestor (LCA) of (*a, b*) is at level 1 while the LCA of (*d, c*) is at the root (level 0). However, if a link exists between node *b* and node *c*, it will be difficult to tell which node pair are topologically closer thus less likely to exhibit worse skew. In this case, we will use the non-tree delay calculation technique (Section 2) to obtain the worst case skew of each sink node pair in order to decide which pairs will likely have large skew.

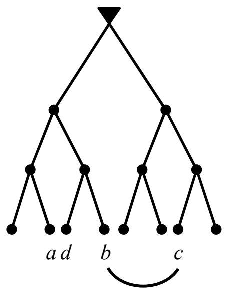

Fig. 2. Clock Network with Links

In our approach, we will insert the first link between the two subtrees of the root. We will find the node pair with the smallest physical distance and thus the smallest resistance. Starting from the second link, we will formulate a sequence of linear programs to find the most beneficial node pair for inserting a link. Fig. 3 shows an overall flow of our algorithm to insert links. In the following section we will discuss in more details of the sequence of linear programs.

```
Algorithm: Add Links
Input: A clock distribution network
Output: $m$ crosslinks
1. Insert the first link between node pair ($a$, $b$) where $a$ and $b$ are in
   the left and right subtree of the root and are closest to each other
   physically
2. $k \leftarrow 1$
3. While $k < m$
4.     For each sink node pair ($u$, $v$)
5.         Construct an LP to find the largest $f_{u,v}$ value
           * $f_{u,v}$ is an objective function described below
6.     Return the node pair with the largest $f_{u,v}$ value for adding a link
7.     $k \leftarrow k + 1$
8. End
```

Fig.3. Link Insertion Overview

3.1 Linear Programs for Selecting Node Pairs

Given a non-tree clock network, we will first make use of the techniques discussed in Section 2 to decompose the network into a tree structure and redistribute the load capacitances. An example is illustrated in Fig. 4(a). In Fig. 4(a), a link *k* is added between sink nodes S_1 and S_2, with load capacitance C_1 and C_2 respectively. First we add half of the link capacitance C_{link} to endpoints S_1 and S_2 of the link. When the tree is decomposed, we pick one of the link endpoints to decompose. In this example, we pick node S_1 to be viewed as being split into two nodes as in Fig. 4(b). The new node S_3 is connected to S_2 through the added link. The new loading capacitance at node S_1 is $C_1 + C_{link}/2 - c_k$ where c_k is the capacitance distributed to node S_2. Suppose that there are m links in the clock distribution network, similar decomposition can be done for all the links. We can use m variables $c_1, c_2, \ldots, c_m$ to describe the load redistribution from one endpoint of a link to the other. Finally, we obtain a tree structure with some way of load distribution. The Elmore delays in the clock network can then be found.

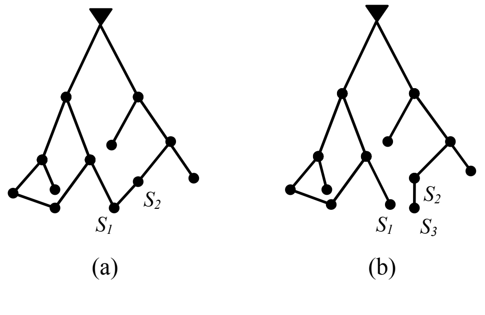

Fig. 4. Decomposition of Clock Network with Links

Once the network is decomposed into trees and the load capacitances are redistributed, according to equation (5), we can write the Elmore delay of each sink node as a linear function of the sink load capacitances. For a buffered clock network, the delay can also be expressed as a linear function of the sink capacitances. Let n be the total number of sinks and m be the number of links, the delay of a sink node S_u is

$$t_u = f_u(C_1, C_2, ..., C_n, c_1, c_2, ..., c_m) \quad (13)$$

The skew between sink node S_u and S_v is

$$q_{u,v} = |t_u - t_v| \quad (14)$$

Then we can formulate the worst case skew between a pair of nodes under variations. However, adding a link between the pair with the worst skew might involve very long route and a significant increase in wirelength. If the length of the link is large, the resistance of the link will be high. Firstly, this will make the clock network not practical. Secondly, a large wirelength will lower the skew reduction effect of a link as discussed in Section 2. In Fig. 2, if we do not consider the resistance of the link, adding a link between node b and node c will be better than adding a link between node a and node b, since node a and node b share some parts of their paths from the source and the delay difference will be smaller. However, if the distance between node b and node c is larger than the distance between node a and node b, it will be difficult to say which link is better. A tradeoff between the length of a link and its clock skew reduction effect is needed. Therefore, when selecting node pairs, we consider both clock skew and link length to find the node pairs which are most beneficial for link insertion. We thus define an objective function to strike a balance between the worst case skew and the link resistance $r_{u,v}$ as follows:

$$f_{u,v} = q_{u,v}/r_{u,v} \quad (15)$$

This objective function is established empirically. From the experimental results, we find that this objective function can achieve low clock skew effectively. We can find the maximum of $f_{u,v}$ subject to changes in the load capacitance values and a set of linear equality constraints as described above.

The sink load capacitances have upper and lower bounds. For each sink node S_i

$$l_i < C_i < u_i \quad (16)$$

We also have a set of linear equality constraints to describe the delay equalities due to the links. Let $S_{left(k)}$ and $S_{right(k)}$ be the two endpoints of link k. W.l.o.g, let $S_{left(k)}$ be the node being split and $S_{left(k)}$ becomes $S_{left(k)}$ and $S_{mid(k)}$ after splitting. Assume that the delays at $S_{left(k)}$ and $S_{right(k)}$ are t_{k1} and t_{k2} respectively, and the delay at $S_{mid(k)}$ is t_{k3}. D^k_{link} is the delay on the link connecting $S_{mid(k)}$ and $S_{right(k)}$. Assume that the link resistance is R^k_{link}, the equality constraints due to the k^{th} link can be written as:

$$t_{k1} = t_{k3} = t_{k2} + D^k_{link} \quad (17)$$

$$D^k_{link} = R^k_{link} c_k \quad (18)$$

To summarize, the optimization problem for maximizing the objective function $f_{u,v}$ between node S_u and S_v is:

Maximize: $f_{u,v} = |t_u - t_v|/r_{u,v}$

Constraints:

$$t_u = f_u(C_1, C_2, ..., C_n, c_1, c_2, ..., c_m) \quad (19)$$

$$t_v = f_v(C_1, C_2, ..., C_n, c_1, c_2, ..., c_m) \quad (20)$$

$$l_i < C_i < u_i \qquad i = 1,...,n$$

$$t_{k1} = t_{k2} + D^k_{link} \qquad k = 1,...,m \quad (21)$$

In our implementation, instead of calling a linear solver, we will first use Gaussian elimination to solve the k equality constraints (21), so the k variables c_i can be expressed by $C_1, C_2, ..., C_n$. Equation (19) and (20) become

$$t_u = f_u'(C_1, C_2, ..., C_n) \quad (22)$$

$$t_v = f_v'(C_1, C_2, ..., C_n) \quad (23)$$

Then the maximum objective value subject to changes in the sink load capacitances can be computed directly from the upper and lower capacitance bounds.

The maximum $f_{u,v}$ (called $p_{u,v}$) for a pair of node S_u and S_v can be obtained by solving the above linear program. Finally, we can find a node pair with the largest $p_{u,v}$. A crosslink is added between this node pair to reduce the clock skew while constraining the wirelength of the link. By applying the technique recursively, we can add a user-defined number of crosslinks to construct a link based clock network.

3.2 Reducing the Number of Optimizations

Although the optimization problem for a pair of nodes can be solved quickly as formulated above, we need to run it for all pairs of nodes to get the node pair with p_{max}. The running time will increase quickly when the number of nodes increases to thousands. Therefore, we devised a method to speed up the process when the number of sinks is large. In order to reduce the running time, instead of trying all pairs of sinks, we will choose some node pairs which are more likely to have the p_{max}. Actually there are some pairs that we do not need to consider, e.g., nodes that are topologically close to each other. This is because these node pairs share a long mutual sub-path from the source and the delay differences between them will not be big.

Starting from the sink node level, we will travel up to find a set of internal nodes I such that the set S of subtrees rooted at these internal nodes should cover all the sink nodes. The detailed procedure to find the internal nodes is shown in Fig. 5. Compared with the total number of sinks, the number of these internal nodes will be smaller. Since the sink nodes within a subtree in S will likely to have similar delays, we will not consider their skews. We will pick one node from each subtree in S to find their skew. In our implementation, we will also consider the number of sinks in each subtree in S and limit them to be less than or equal to 16. In this way, even benchmarks with a large number of sinks can be solved in a reasonable amount of time.

```
Algorithm: Find Nodes
Input: A clock network
Output: Internal node set $I$
1. $I = \Phi$
For each sink node $S_i$
2. If $S_i$ is contained in a subtree rooted at a node in $I$
3.     Continue
4. Else
5.     $k \leftarrow 1, r \leftarrow 1, I_i \leftarrow S_i$
6.     While $r <$ lower bound in the sink number of a subtree
7.         $k \leftarrow k + 1, I_i \leftarrow$ parent of $I_i$, temp $\leftarrow I_i$
8.         $r \leftarrow$ number of sinks contained in the subtree rooted at $I_i$
9.     End
10.    If $r >$ upper bound in the sink number of a subtree
11.        $I_i \leftarrow$ temp
12. Add $I_i$ to $I$
```

Fig. 5. Find a Set of Internal Nodes I

4. EXPERIMENTS

Our algorithm to select node pairs for non-tree clock network construction with crosslinks is implemented in C. The experiments are performed on a 3.2 GHz Intel CPU Linux machine with 4GB memory. To verify the effectiveness of our method, we will compare our link insertion scheme with the work in [1] and the work in [4]. These works did not consider non-tree delay when inserting links. We implemented the minimum weight matching-based link insertion method which is the best method in [1]. We start with the same zero skew clock tree which is obtained from the bounded skew tree method [2]. The benchmarks and the bounded skew tree code are downloaded from the GSRC bookshelf (http://vlsicad.eecs.umich.edu/BK/). For comparison with the work in [4], experiments are performed on the ISPD 2010 contest benchmark. In our implementation and simulations, the per unit wire resistance is 10^{-4} Ω/nm and the per unit wire capacitance is 2×10^{-4} fF/nm. *Ngspice* is used to simulate our clock distribution network. Process variations are accounted in the simulations for VDD variations and sink capacitance variations. We allow 15% variations in the VDD and the sink capacitances and all the variations follow a normal distribution. For each clock network, 500 *spice* simulations are performed to obtain the worst case skew (WCS).

All the benchmark information and the experimental results of our work and the method in [1] are shown in Table 1. The benchmark sizes, worst case skews (WCS), and the wire capacitances of the clock trees are given. We list the number of links inserted, WCS results, link capacitance results of the Link-M method and our work. The Link-M method refers to the minimum weight matching based link insertion method in [1]. From the table, we observe that our method always outperforms the Link-M method in terms of clock skew reduction. Besides, for a same number of links, Link-M method costs 249% link capacitance comparing with ours on average. The reason is that the Link-M method does not consider non-tree delay when inserting links, which may result in ineffective crosslink insertion into the clock tree. By considering the balance between clock skew and link resistance, we can achieve more clock skew reduction, and the link capacitance can also be reduced. We list the CPU time in seconds in the table. The CPU time for the Link-M method is ignorable so it is not listed in the table. The technique discussed in Section 3.3 can be used to control the running time and we can see the running time is acceptable.

To compare our proposed method with the most recent work [4] which inserts links in the internal nodes of a tree and insert links while constructing the tree, we obtain the clock network results from the authors of [4]. The links added between the internal nodes are removed before our method is applied to generate crosslinks into the clock network. Table 2 compares our work with the method in [4]. LCS refers to the local clock skew constraint on the benchmark in ISPD 2010 High Performance Clock Network Synthesis Contest [15]. Our work uses less link resources to achieve similar LCS results. Note that the CPU time of our work in the table is for the link insertion phase only, while the tree construction time is not included. Please note that the two results are not directly comparable, because in [4], the links are inserted while the trees are being constructed, so their tree construction performs in such a way to favor the link insertion. In our case, we take their trees and insert the links in a post processing way, so the results are hard to be compared. However, we still want to display the comparisons to show that our method can also handle the clock trees with buffer and can perform actually quite well comparing with [4] in which the link insertion and the tree construction are performed simultaneously.

5. CONCLUSION

In this paper, signal delay and clock skew in non-tree clock networks are discussed. Based on the analysis, a new method is proposed to select node pairs for link insertion in a clock network. Experimental results show that this method can be applied to insert links effectively.

6. REFERENCES

[1] A. Rajaram, J. Hu, and R. Mahapatra. Reducing clock skew variability via crosslinks. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 25, no. 6, 2006, 1176–1182.

[2] J. Cong, A. B. Kahng, C.-K. Koh, and C.-W. A. Tsao. Bounded-skew clock and steiner routing under Elmore delay. In *Proceedings of the 1995 IEEE/ACM International conference on Computer-aided design*, 1995, 66–71.

[3] P. Chan and K. Karplus. Computing signal delay in general RC networks by tree/link partitioning. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 8, 1990, 898–902.

[4] T. Mittal and C.-K. Koh. Cross link insertion for improving tolerance to variations in clock network synthesis. In *Proceedings of the 2011 international symposium on Physical design,* 2011, 29–36.

[5] P. Restle, T. McNamara, D. Webber, P. Camporese, K. Eng, K. Jenkins, D. Allen, M. Rohn, M. Quaranta, D. Boerstler, *et al.*. A clock distribution network for microprocessors. *IEEE Journal of Solid-State Circuits*, vol. 36, no. 5, 2001, 792–799.

[6] G. Venkataraman, Z. Feng, J. Hu, and P. Li, Combinatorial algorithms for fast clock mesh optimization. In *Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, 2006, 563–567.

[7] H. Chen, C. Yeh, G. Wilke, S. Reddy, H. Nguyen, W. Walker and R. Murgai. A sliding window scheme for accurate clock mesh analysis. In *Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, 2005, 939–946.

[8] A. Rajaram and D. Z. Pan. MeshWorks: An efficient framework for planning, synthesis and optimization of clock mesh networks. In *Proceedings of the 2008 Asia and South Pacific Design Automation Conference*, 2008, 250–257.

[9] W. -C. D. Lam, J. Jain, C. -K. Koh, V. Balakrishnan, and Yiran Chen. Statistical based link insertion for robust clock network design. In *Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, 2005, 588-891.

[10] G. Venkataraman, N. Jayakumar, J. Hu, P. Li, S. Khatri, A. Rajaram, P. McGuinness, and C. Albert. Practical techniques for minimizing skew and its variation in buffered clock networks. In *Proceedings of the 2005 IEEE/ACM International conference on Computer-aided design*, 2005, 592–596.

Table 1. Benchmark information, worst case skew and wire cap results with 15% variations in vdd and sink capacitances

Bench -mark	#Sinks	WCS (ps)	Wire Cap (fF)	#Links	Link-M WCS (ps) a (a/b)	Our WCS (ps) b	Link-M Link Cap (fF) c (c/d)	Our Link Cap (fF) d	CPU (s)
r1	267	0.364	264.1	4	0.314 (1.33)	0.236	9.7 (3.24)	2.9	0.81
				16	0.162 (1.41)	0.115	16.3 (1.23)	13.2	3.21
				22	0.122 (1.20)	0.102	60.0 (3.14)	19.1	4.52
r2	598	0.802	523.7	4	0.772 (1.68)	0.460	6.9 (4.93)	1.4	9.23
				20	0.286 (1.39)	0.206	21.0 (1.60)	13.1	46.47
				40	0.192 (1.42)	0.135	40.7 (1.46)	27.9	94.09
r3	862	1.250	678.2	6	0.470 (1.09)	0.433	10.2 (1.96)	5.2	2.74
				24	0.305 (1.08)	0.282	25.9 (1.07)	24.3	11.73
				48	0.229 (1.36)	0.168	101.6 (1.91)	53.2	27.21
r4	1903	2.054	1357.8	8	1.565 (1.55)	1.008	41.6 (8.67)	4.8	9.21
				40	0.448 (1.21)	0.370	57.3 (1.23)	46.5	64.34
				68	0.336 (1.03)	0.327	125.3 (1.42)	88.3	140.3
r5	3101	2.612	2005.5	8	1.451 (1.14)	1.276	43.2 (7.20)	6.0	36.17
				40	0.740 (1.14)	0.648	76.2 (1.67)	45.7	229.8
				72	0.589 (1.16)	0.506	95.9 (1.08)	89.0	510.5
s1423	74	0.048	21.30	4	0.042 (1.83)	0.023	1.33 (1.17)	1.14	0.03
				8	0.036 (2.57)	0.014	2.77 (1.07)	2.60	0.03
s5378	179	0.061	35.10	4	0.063 (1.75)	0.036	1.60 (2.86)	0.56	0.29
				12	0.025 (1.09)	0.023	5.94 (2.57)	2.31	0.89
s15850	597	0.136	89.65	4	0.111 (1.34)	0.083	0.56 (1.27)	0.44	10.62
				22	0.053 (1.39)	0.038	3.88 (1.56)	2.48	59.82

Table 2. Comparison of our method with the work in [4]

Bench -mark	#Sinks	Method	LCS (ps)	Ratio of Link Cap	CPU(s)
01	1107	[4]	7.88	1.007	1092
		Our work	10.67	1	52.4
02	2249	[4]	8.32	1.223	4314
		Our work	8.17	1	424
03	1200	[4]	6.34	1.073	383
		Our work	5.89	1	10.6
04	1845	[4]	7.42	1.001	934
		Our work	7.33	1	33.8
05	1016	[4]	5.90	1.199	278
		Our work	5.87	1	5.36
06	981	[4]	6.78	1.003	285
		Our work	8.98	1	6.69
07	1915	[4]	6.77	1.228	818
		Our work	6.05	1	49.8
08	1134	[4]	6.42	1.225	327
		Our work	8.59	1	8.34

[11] Anand Rajaram and David Z. Pan. Variation tolerant buffered clock network synthesis with cross links. In *Proceedings of the 2006 international symposium on Physical design*, 2006, 157–164.

[12] Z. Li, Y. Zhou, and W. Shi. Wire sizing for non-tree topology. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 26, no. 5, 2007, 872–880.

[13] T. M. Lin and C. A. Mead. Signal delay in general RC networks. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 3, no. 4, 1984, 331–349.

[14] P. K. Chan and M. D. F. Schlag. Bounds on signal delay in RC mesh networks. *IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems*, vol. 8, no. 6, 1989, 581–589.

[15] http://archive.sigda.org/ispd/contests/10/ispd10cns.html.