Local Clock Skew Minimization Using Blockage-aware Mixed Tree-Mesh Clock Network

Linfu Xiao, Zigang Xiao, Zaichen Qian, Yan Jiang, Tao Huang, Haitong Tian and Evangeline F.Y. Young
Department of Computer Science and Engineering, The Chinese University of Hong Kong, Shatin, N.T., Hong Kong
email: {lfxiao, zgxiao, zcqian, yjiang, thuang, httian, fyyoung}@cse.cuhk.edu.hk

***Abstract*— Clock network construction is one key problem in high performance VLSI design. Reducing the clock skew variation is one of the most important objectives during clock network synthesis. Local clock skew (LCS) is the clock skew between any two sinks with distance less than or equal to a given threshold. It is defined in the ISPD 2010 High Performance Clock Network Synthesis Contest [1], and it is a novel criterion that captures process variation effects on a clock network. In this paper, we propose a hybrid method that creates a mesh upon a tree topology. Total wire and buffer capacitance is minimized under the LCS and slew constraints. In our method, a clock mesh will be built first according to the positions and capacitance of the sinks. A top-level tree is then built to drive the mesh. A blockage-aware routing method is used during the tree construction. Experimental results show our efficiency and the solution generated by our approach can satisfy the LCS constraint of all the benchmarks in the contest [1], with a fair capacitance usage.**

I. Introduction

A. Clock Network Synthesis

Clock network construction is one of the most important research topics in industrial and academic communities. Among the possible objectives including power, signal slew, nominal skew and wire length, the nominal skew is a crucial one because it directly relates to the maximum frequency of the circuit. The unwanted clock skew may be caused by variations like buffer manufacturing variation, power-ground noise, etc. and contributes to a large portion of the nominal skew. Robust clock network that is less sensitive to process variation is highly desired.

B. Previous Works

In the literature, the approaches to this clock network synthesis problem can be classified into two categories, i.e., clock tree and clock mesh. An example of a clock tree and a mesh structure is given in Fig. 1(a) and Fig 1(b) respectively. Clock tree is a traditional method that is commonly used in commercial tools. Basic methods include H-tree [2], the Method of Means and Medians (MMM) [3], Path Length Balancing method (PLB) [4] and Deferred Merge Embedding (DME). DME was first introduced in [5], [6], [7], and there have been some variants of DME devised like Planar-DME [8]. This type of clock distribution has the advantages of low power consumption and shorter wire length, as well as simplicity of analysis, implementation and simulation. However, it is prone to have clock skew when variation exists. Clock mesh on the other hand, provides better tolerance due to the existence of multiple paths from the clock source to the sinks [9]. Nevertheless, significantly higher wire area will be consumed in a clock mesh, which leads to a higher power dissipation. The authors of [10] and [11] proposed some methods to analyze the characteristics of a mesh network. Venkataraman *et al.* proposed a combinatorial algorithm in [9] to optimize a clock mesh to trade-off between power and tolerance to process variation.

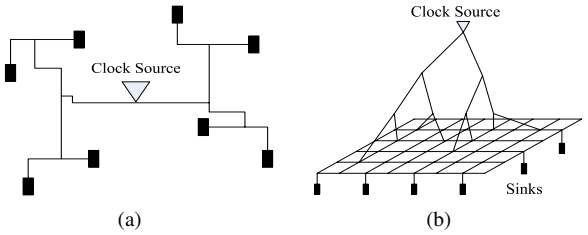


Fig. 1. (a) Tree structure clock network. (b) Mesh structure clock network.

Hybrid structure that combines tree and mesh also exists, e.g., [12], [13], [14]. This type of structure is a compromise between tree and mesh, but the mixed structure makes the analysis extremely difficult. Apart from tree and mesh style networks, a technique named link insertion [15], [16] is also developed and can be used in the aforementioned structures, which will change a clock tree to a non-tree structure. By using this technique, cross links will be inserted between unbalanced sinks in order to reduce their skew.

C. Local Clock Skew

To promote the clock network research, a clock network synthesis contest [1] is launched recently that focuses on the process variation issue. In this contest, Local Clock Skew (LCS) is defined as the clock skew between any two sinks with a distance less than or equal to a given threshold (e.g., $600nm$) [1]. Let s_i and s_j be two clock sinks. $T(s_i)$ denotes the clock arrival time of s_i and $MHT(s_i, s_j)$ denotes the Manhattan distance between two points s_i and s_j. L is a given threshold. We have:

$$LCS = \max_{MHT(s_i,s_j)\leqslant L} \{\ |T(s_i) - T(s_j)|\ \} \tag{1}$$

In the contest, LCS inevitably exists because of the variation in buffers and wires. It is considered to be a more practical metric to be optimized, as nearby points on a chip are the primary ones that are susceptible to short-path errors caused by clock skew [12], and it is also being used in industry [17].

D. Our Contributions

In this paper, we propose a blockage-aware mixed tree-mesh clock network to reduce local clock skew caused by process variation. In our framework, an irregular mesh will first be generated according to the positions and capacitance of the sinks. A set of *grid points* are then generated and distributed in the layout region. The sinks and the corresponding nearest grid point will be connected using a rectilinear minimum spanning tree. *Grid links* are inserted between grid points so as to reduce the skew caused by process variation. We will treat these grid points as sinks in our extended-DME algorithm to build a balance tree. Our method is fast and scalable to handle large data set and tight LCS constraint. Besides, we provide some analysis for the proposed mixed tree-mesh clock network.

TABLE I

COMPARISON BETWEEN TWO APPROACHES

	Mean	Stdv*	Max	#Violations
Method (a)	16.77	3.64	24.19	50
Method (b)	4.45	1.09	6.88	0

*Standard deviation.

The remainder of this paper is organized as follows. Section II gives a detailed description and formulation of the problem. Section III introduces our approach. An analysis of the mixed tree-mesh structure will be given in Section IV. Section V reports the experimental result and comparison. Our conclusion is drawn in Section VI.

II. PROBLEM FORMULATION

A. Clock Network Synthesis Problem

In this paper, the problem is modeled based on the specification of the ISPD 2010 High Performance Clock Network Synthesis Contest [1]. A formal description of the problem is given as follows: The inputs are (1) a $W \times H$ layout region, (2) a set of n sinks and m blockages and (3) a library of buffers and wires. Details can be found in [1]. We want to obtain a clock network with non-inversed output that satisfies all the constraints, while minimize the total buffer and wire capacitance. The constraints are (1) a *Local Clock Skew* limit L within a threshold distance D and (2) a slew limit T.

Besides, we assume that buffers cannot be placed on a blockage. However, routing can be done over a blockage. An open source circuit simulator, ng-spice, will be used to simulate the constructed clock distribution network. Note that Monte Carlo simulations will be conducted to evaluate a clock network under different buffer Vdds and wire variations.

B. Effects of Hypothetical Parallel Buffers

In the contest [1], buffers are allowed to be added in parallel at the same place. Besides, on-chip variations upon wires and buffers are assumed to be independent. In particular, there will be a +/-7.5% variation of the supply voltage for each buffer independently. Hence, one can utilize parallel buffers to cancel out the process variation effects. We verify our statement by performing an experiment. In this experiment, the layout region contains 16×16 evenly distributed sinks. An H-tree will be generated first, and the buffer locations in the tree are also computed. Then, we add buffers in the two following ways to construct two different clock trees. For each buffer location:

(a) Only one type-1 buffer will be added.
(b) Ten type-2 buffers will be connected in parallel to provide similar driving strength as one type-1 buffer.

After the trees are constructed, fifty simulations will be performed for each of them. As shown in Table I, method (a) obtained a 4x times worse average LCS and 3x times worse standard deviation comparing with method (b). At the same time, method (a) exceeds the LCS constraint in all simulations, while method (b) can always satisfy the constraint. This demonstrates the effects of parallel buffers in this problem. However, parallel buffers and variation independence are impractical and inaccurate. Correlation usually exists between closely placed buffers after fabrication. Their electronic characteristics tend to be similar and dependent on each other. In our proposed approach, we will add only one buffer at any specific location.

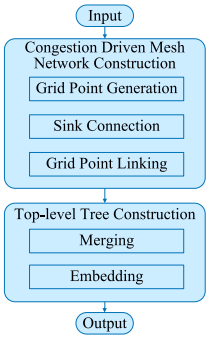


Fig. 2. Overview of our approach.

III. BLOCKAGE-AWARE TREE-MESH CLOCK NETWORK CONSTRUCTION

A. Overview of Our Methodology

Our clock network construction is divided into two stages (Fig. 2). The first stage is Congestion Driven Mesh Network Construction. According to the congestion and capacitance information of the sinks, a set of *grid points* will be generated and distributed across the layout region. For each grid point, a *minimum spanning tree* (MST) will be constructed to connect the nearby sinks to it. *Grid links* are then inserted systematically between selected grid points to increase the stability of the network. A top-level tree is then utilized to drive the grid points after the sinks are attached to the grid points. Instead of the original sinks, the grid points will be viewed as new sinks in the second stage of tree construction. An extended DME algorithm that uses a *minimum weight matching* in pair selection is used to build a tree structure. During this stage, a blockage-aware routing is performed. Here we adopt a blockage-aware scheme rather than a blockage-avoiding scheme. In particular, we allow a route to cross a blockage, as long as the buffers can be placed correctly outside the blockage and the slew limit is not violated. This strategy is more sophisticated than just avoiding the blockages, and can provide us a more flexible routing solution. In the following sections, each component of our algorithm will be presented.

B. Congestion Driven Mesh Network Construction

At this stage, we want to generate a set of grid points to form a mesh. This step will be done recursively. Initially, a coarse mesh will be constructed. It will be recursively divided according to the distribution of the sink capacitance. We call each individual rectangle in the mesh a *room*, and the corners of the rooms *grid points*. A room in the mesh will be divided recursively into 2×2 if the estimated total capacitance of the sinks and wires inside is larger than a threshold. The division step repeats until the capacitance estimation is small enough in each room, or there is only one sink in each room. After this division step, the corners of the rooms will be used as grid points and sinks will be connected to their nearest grid points. To reduce wire length, a rectilinear minimum spanning tree will be used to connect a grid point with its nearby sinks.

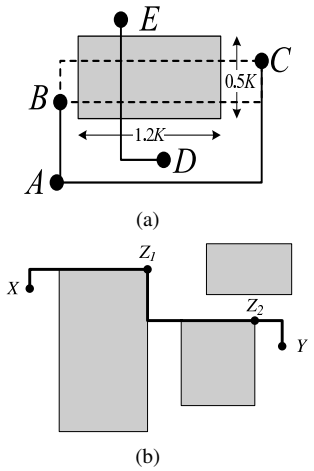


Fig. 3. (a) An example of *connectivity*. (b) An example of a shortest path between X and Y generated by our method.

At last, *grid links* will be inserted between horizontally or vertically adjacent grid points to form the final mesh.

C. Top-level Tree Construction

At this stage, grid points generated in the previous stage will be treated as sinks and they are the leaves of the DME tree. The capacitance of each of these sinks is the sum of the capacitance contributed by all the original sinks connected to it and the corresponding wires. An extended-DME algorithm follows to build a top-level tree to drive these grid points. It contains a bottom-up *blockage-aware merging* phase and a top-down *embedding* phase. In the bottom-up merging step, pairs of sinks will be selected to *merge*. A *merging segment* will be created for each pair, and the merging point will be viewed as a sink at the next level. We call the nodes in a DME tree the *DME nodes*. We will denote a DME node by a segment $\overline{XY}$, where X and Y are its endpoints meaning that the actual location of the DME node can be anywhere on the segment. For the pair selection step, a *minimum weight matching* algorithm (see Section III-C.4) is used to minimize the total wire length at each level of the tree. After the topology of the tree is computed, a top-down *embedding* is performed to finalize the actual position of each tree node. Elmore delay model is used to calculate the buffer locations and merging point. During this phase, buffers will be inserted regularly to prevent slew degradation. Note that due to the existence of blockages, several parts of a merging segment may locate inside a blockage. Those parts inside a blockage will be removed to avoid inserting buffers on the blockage. In the following sub-sections, we will explain our modification over the original DME algorithm in detail.

1) ***Blockage-aware Routing****:* we will introduce our routing method first. In our approach, buffers will be inserted in the top-level clock tree regularly in order to maintain a valid slew and they cannot overlap with the blockages. Here we will use a *dynamic blockage-sink rectilinear graph* G to perform routing. The vertex set V of the graph G consists of the current sinks, the corners of the blockages and the merging points (internal nodes of the clock tree). A connectivity test will then be performed to determine the edge set E in G. The connectivity of two vertices is defined as follows:

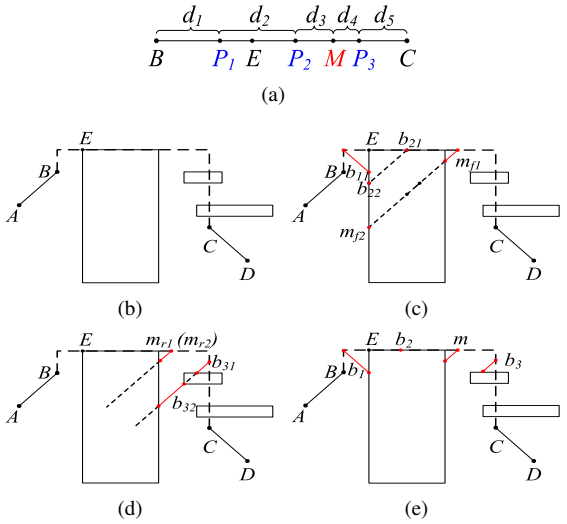


Fig. 4. An example of blockage-aware merging.

Definition *1 (Connectivity of two vertices):* Let u, v be two vertices in G. We say that u and v are *connected* if there exists a zero-bend or a one-bend rectilinear path p between u and v satisfying:

- p is not blocked by any blockage, or
- p is blocked by some blockages b_1, b_2, ..., b_n horizontally/vertically, but the width/height of the blockages are smaller than the buffer insertion distance.

The edge weight between u, v equals to the length of p. An example is given in Fig. 3(a). Suppose that the buffer insertion distance is K. The sinks $(A, B), (A, C), (E, D)$ are *connected*, while sinks (B, C) is *disconnected*. Then, a rectilinear shortest path between any two points can be obtained from G by applying a shortest path algorithm (e.g., $Dijkstra$) on it. A typical shortest path constructed by our method is illustrated in Fig. 3(b) in which $X - Z_1 - Z_2 - Y$ is the path.

2) ***Blockage-aware Merging and Embedding****:* We propose a concept of *buffer segment* that is similar to the merging segment to provide alternative insertion places for buffers. A *buffer segment* of a DME node n is a set of possible buffer locations that have the same Manhattan distance from n. This idea is inspired and extended from a previous work [18]. We consider this buffer segment as early as in the merging stage. By doing this, more valid insertion places can be found for a buffer even when the wire is routed over some blockages. Although this buffer segment approach may also encounter cases in which a segment locates inside a blockage completely, the mark-and-reroute method can be used to resolve this problem. We define *Rectilinear Shortest Path* between two segments as follows:

Definition *2 (Rectilinear shortest path (RSP)):* For two lines $\overline{AB}$ and $\overline{CD}$, the *rectilinear shortest path* between them are the rectilinear shortest path between two nearest points in $\overline{AB}$ and $\overline{CD}$ respectively, and denoted as $RSP(\overline{AB}, \overline{CD})$.

In the following, we illustrate our merging algorithm by giving an example. In Fig. 4(b), two DME nodes $\overline{AB}$ and $\overline{CD}$ are selected for merging. The rectilinear shortest path between them is then computed as $RSP(\overline{AB}, \overline{CD}) = B - E - C$, where E is an intermediate blockage corner node. Let l be the length of this path. In order to compute the merging segment and buffer segments, the balance point and buffer points will be determined first. This will be discussed in

Section III-C.3. In this example, three buffers will be inserted and we assume that the distances between these buffer and balance points are as shown in Fig. 4(a), where M is the balance point, and P_1, P_2 and P_3 are the buffer points. A merging segment m corresponding to M and three buffer segments b_1, b_2 and b_3 corresponding to P_1, P_2 and P_3 will be computed. A *forward and reverse* two-way computation is used to determine these segments. The forward one computes the segment candidates for b_1, b_2 and m_f, while the reverse one computes the segment candidates for b_3 and m_r. The result of the forward and reverse computation is shown in Fig. 4(c) and Fig. 4(d) respectively. The solid lines are buffer or merge segment candidates, while the dash lines are those parts removed because of the blockages. Because in forward direction ($AB-m_f$), we do not consider the blockages within $CD-m_r$ path, and vice versa. Hence, a longest intersection between $\{m_{f1}, m_{f2}\}$ and $\{m_{r1}, m_{r2}\}$ is then computed to determine the final m. Finally, We trace back to select all the buffer segments for b_1, b_2 and b_3 (see Fig. 4(e)).

Embedding is then performed to decide the actual locations of the merging points in their corresponding merging segments, and the actual buffer locations in their corresponding buffer segments. This step is the same as in the original DME algorithm.

3) ***Buffer Insertion****:* During the merging step, balance point and buffer locations will be computed. The buffers will be added regularly in our top-level clock tree to maintain the slew within the limit. The buffer insertion distance is determined empirically. Based on the Elmore delay model, we use a trial-and-error method to determine the number of buffers. In particular, the number of buffers will be enumerated iteratively. Furthermore, as mentioned in Section II-B, we will add only one buffer at each buffer location in order to be more practical.

4) ***Minimum Weight Matching****:* During the merging phase, to decide the topology of the DME tree, a graph will be constructed at each level of the tree for matching. All the current DME nodes existing in the layout region will be represented as vertices in this graph. The edge weight is computed as follows:

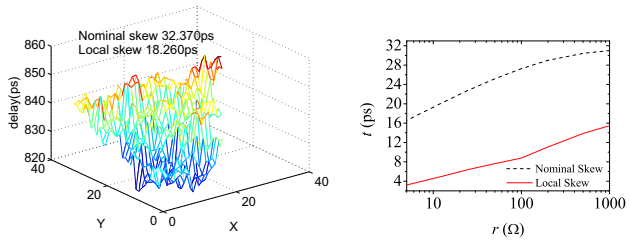
$$w(e) = MHT(i,j) + \sqrt{\frac{|(t_i + cap_i \cdot R_B) - (t_j + cap_j \cdot R_B)|}{\alpha_w \beta_w}} \quad (2)$$

where t_i and cap_i are the downstream delay and capacitance of node i, R_B is the output resistance of a buffer, α_w and β_w are the unit wire resistance and capacitance of a wire. Note that there is no LCS constraint for two sinks that have a distance greater than D. Hence we will set the weight of this edge ∞.

Under this scheme, the topology resulted will be a *forest* that contains multiple DME trees. We will then use an obstacle-avoiding rectilinear Steiner minimum tree (SMT) [19] to connect the roots of all these trees with the clock source.

IV. Analysis

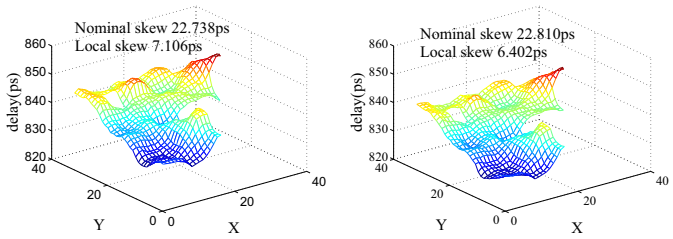
In this section, we will provide some analysis for the proposed mixed tree-mesh clock network. Our goal is to understand more how a mesh structure can help to reduce the skew. The analysis is based on the benchmark ispd10cns01 [1]. A 33×33 mesh is used for the analysis. We use both ng-spice and the computational method proposed in [20] to compute delays.



(a) (b)

Fig. 5. (a) The delay map of the pure tree. (b) The relationship between r and the local clock skew.

Fig. 5(a) shows the delay map of the pure tree, i.e., the top-level tree without mesh. Fig. 6 shows the delay maps of the proposed mixed tree-mesh network with one grid link inserted between each pair of adjacent grid points. The sub-figure on the left is obtained by simulation while the one on the right is obtained by the computational model. Comparing these three figures, we can see that the clock skew can be reduced significantly by using the proposed mixed tree-mesh network.



(a) (b)

Fig. 6. (a) Delay map of the network with one link (simulation). (b) Delay map of the network with one link (computation).

We can also see by comparing the two sub-figures in Fig. 6 that the results of the ng-spice simulation are very similar to the results of the recursive computational model. The correlation coefficients between the two delay results are over 0.98, which indicate that the computational model in [20] can describe the empirical delay values fairly accurately. Therefore, we can make use of this model to further analyze the mixed tree-mesh network. By setting different values to the grid link resistance r, we can derive the relationship between the grid link resistance and the local clock skew as shown in Fig. 5(b). We can observe that the LCS is almost proportional to the exponential of r. By adding more links in parallel or using a finer mesh, we are able to achieve a better local clock skew because the grid link resistance is reduced.

V. Experimental Result

The proposed algorithm is implemented in C. The experiment is carried out on an 3.20GHz Intel CPU Linux workstation with 2GB memory. Benchmarks from the contest [1] are used to evaluate the solution quality. Monte Carlo simulation provided in the evaluation script in the contest is performed to simulate the process variation on Vdd and the wire variations. Our solution for each benchmark will be simulated for a

TABLE II

COMPARISON ON RUNNING TIME, LCS AND CAPACITANCE

B.N.[†]	Our Approach (no parallel buffers)								Contango (1st place, tree, 30 type-2 buffers in parallel)						
	CPU(s)[‡]	Mean	95%	Max	Cap[‡]	Inv-c	Tree-c	Mesh-c	CPU(s)[‡]	Mean	95%	Max	Cap[‡]	Inv-c	Tree-c
01	675/1.0	5.07	7.23	9.43	1168104/1.0	530035	146227	472920	12015/17.8	5.16	7.01	10.11	198337/0.2	117782	61633
02	2140/1.0	5.43	7.35	8.99	2099811/1.0	847435	239661	973495	25006/11.7	5.58	7.34	9.24	375863/0.2	227869	108775
03	21/1.0	2.89	3.95	4.23	93965/1.0	49335	16881	9060	3840/182.9	3.03	4.18	5.04	55861/0.6	31097	6615
04	22/1.0	5.27	7.25	7.64	125333/1.0	65090	31102	16795	6075/276.1	3.26	4.46	6.06	71843/0.6	40326	19170
05	10/1.0	4.34	7.27	9.40	74084/1.0	39100	19409	10349	1383/138.3	25.19	26.91*	25.58	35496/0.5	19891	10378
06	46/1.0	5.75	6.79	8.04	87390/1.0	45080	20250	9485	545/11.8	23.93	24.63*	25.22	45719/0.5	25824	7319
07	27/1.0	4.18	5.97	6.67	128351/1.0	65665	28653	15935	2351/87.1	3.41	4.58	5.61	72664/0.6	40326	14239
08	18/1.0	3.58	5.37	6.62	97421/1.0	50255	21512	12609	682/37.9	11.54	12.71*	13.66	51515/0.5	29779	8692
	CNSRouter (2nd place, mesh+tree, 4 type-1 buffers in parallel)								NTUclock (3rd place, tree, 20 type-2 buffers in parallel)						
01	20/0.030	5.27	7.32	10.74	841207/0.7	489095	333190	-	15/0.022	6.71	8.66*	11.14	293887/0.3	159434	115532
02	81/0.038	6.27	8.33*	10.83	1E+06/0.5	819375	594334	-	176/0.082	8.27	10.73*	14.44	832483/0.4	558234	235029
03	8/0.381	1.43	2.70	3.55	162570/1.7	81995	62427	-	6/0.286	6.82	8.63*	10.27	167062/1.8	104629	44284
04	32/1.455	2.83	3.98	5.74	277151/2.2	177215	87590	-	58/2.636	7.45	9.55*	11.36	325206/2.6	181649	131211
05	7/0.70	2.88	4.38	6.51	175580/2.4	120635	49719	-	11/1.10	5.30	6.98	9.79	130389/1.8	65156	60006
06	11/0.239	12.38	14.03*	15.12	133312/1.5	83375	37361	-	10/0.217	406.9	416.62*	421.29	2E+06/22.9	98619	1465951
07	45/1.667	12.39	15.74*	19.62	309242/2.4	201135	90009	-	66/2.444	6.17	8.12*	11.29	295597/2.3	130046	127453
08	19/1.05	6.15	7.33	9.21	222786/2.3	147315	62427	-	7/0.389	5.94	7.64*	9.05	165883/1.7	42484	110355

[†]Benchmark name. To save space, we use '01' for ispdcns01, and so on. [‡]The right part is a normalized value. *LCS violation.

hundred times, except for the first two benchmarks, which will only be simulated for seventy times because of the long simulation time due to their huge sizes. Comparison is made between the clock network generated by our approach and the three winning teams in the contest.

The benchmark details are available in [1]. Table II shows the comparison of the CPU time, statistics on LCS and load capacitance. Note that our solutions employ only one buffer at any specific location. Our algorithm is very efficient and can generate a solution within one minute, except for the two largest cases. For the LCS comparison, note that the contest uses the 95th percentile of the simulation results as the metric to evaluate the worst local clock skew. The '95%', mean and max of LCSs are also shown in the table (in ps). It can be seen that the solutions generated by our approach can satisfy the LCS constraints for all benchmarks, while none of the winning teams can route successfully for all. The total capacitance (cap), inverter capacitance (inv-c), tree capacitance (tree-c) and mesh capacitance (mesh-c) are also given in Table II. Our approach uses on average a capacitance of almost 2x that of the team Contango, which has the lowest capacitance usage among all three teams. Our result is reasonable because we use a mixed tree-mesh structure instead of just one single tree. Besides, unlike other methods, we did not employ any parallel buffers to cancel out the variation effects.

VI. CONCLUSION

We presented a blockage-aware mixed tree-mesh clock network construction algorithm in this paper. The clock distribution network generated by our method is very robust against process variation in minimizing the local clock skew (LCS). The proposed method is scalable and can handle large data sets and tight LCS constraints. By changing the granularity of the mesh or adding more grid links in parallel, the LCS can be further reduced. According to the experimental result, our approach is superior in terms of running time and can satisfy the local skew constraints for all benchmarks.

REFERENCES

[1] ISPD 2010 high performance clock network synthesis contest. ACM SIGDA and Intel Corporation. [Online]. Available: *http://www.sigda.org/ispd/contests/10/ispd10cns.html*

[2] H. Bakoglu, J. Walker, and J. Meindl, A symmetric clock distribution tree and optimized high-speed interconnects for reduced clock skew in ULSI and WSI circuits, *Proc. IEEE Int. Conf. Computer Design*, 1986, pp. 118–122.

[3] M. A. B. Jackson, A. Srinivasan, and E. S. Kuh, Clock routing for high-performance ics. *DAC 90: Proceedings of the 27th ACM/IEEE Design Automation Conference*. ACM, 1990.

[4] A. Kahng, J. Cong, and G. Robins, High-performance clock routing based on recursive geometric matching, *DAC '91: Proceedings of the 28th ACM/IEEE Design Automation Conference*. 1991, pp. 322–327.

[5] K. Boese and A. Kahng, Zero-skew clock routing trees with minimum wirelength, *ASIC Conference and Exhibit, 1992., Proceedings of Fifth Annual IEEE International*, 1992.

[6] T. Chao, J. Ho, and Y. Hsu, Zero skew clock net routing, *Proceedings of the 29th ACM/IEEE Design Automation Conference*. IEEE Computer Society Press, 1992, p.523.

[7] M. Edahiro, Minimum skew and minimum path length routing in VLSI layout design,*NEC research & development*, vol. 32, no. 4, pp. 569–575, 1991.

[8] A. Kahng and C. Tsao, Planar-DME: A single-layer zero-skew clock tree router, IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 1996.

[9] G. Venkataraman, Z. Feng, J. Hu, and P. Li, Combinatorial algorithms for fast clock mesh optimization, *ICCAD '06: Proceedings of the 2006 IEEE/ACM international conference on Computer-aided design*, 2006, pp. 563–567.

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

[11] X. Ye, P. Li, M. Zhao, R. Panda and J. Hu Analysis of large clock meshes via harmonic-weighted model order reduction and port sliding, Proceedings of the 2007 IEEE/ACM international conference on Computer-aided design, 2007.

[12] 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, pp. 792–799, 2001.

[13] H. Su and S. S. Sapatnekar, Hybrid structured clock network construction, *Proceedings of the 2001 IEEE/ACM international conference on Computer-aided design*. 2001.

[14] M. Mori, H. Chen, B. Yao, and C.-K. Cheng, A multiple level network approach for clock skew minimization with process variations,*ASP-DAC '04: Proceedings of the 2004 Asia and South Pacific Design Automation Conference*, 2004, pp. 263–268.

[15] A. Rajaram, J. Hu, and R. Mahapatra, Reducing clock skew variability via cross links, DAC '04: Proceedings of the 41st annual Design Automation Conference, 2004, pp. 18–23.

[16] J.-S. Yang, A. Rajaram, N. Shi, J. Chen, and D. Z. Pan, Sensitivity based link insertion for variation tolerant clock network synthesis, ISQED '07: Proceedings of the 8th International Symposium on Quality Electronic Design, 2007, pp. 398–403.

[17] C. Sze, ISPD 2010 high performance clock network synthesis contest: benchmark suite and results, *Proceedings of the 19th international symposium on Physical design*, 2010.

[18] X.-W. Shih, C.-C. Cheng, Y.-K. Ho, and Y.-W. Chang, Blockage-avoiding buffered clock-tree synthesis for clock latency-range and skew minimization, *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, 2010.

[19] L. Li and Evangeline F.Y. Young, Generation of optimal obstacle-avoiding rectilinear Steiner minimum tree, *Proceedings of the 2009 International Conference on Computer-Aided Design*, 2009.

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