Timing–Driven Variation–Aware Nonuniform Clock Mesh Synthesis

Ameer Abdelhadi, Ran Ginosar, Avinoam Kolodny, and Eby G. Friedman
Dept. of Electrical Engineering
Technion – Israel Institute of Technology
Haifa 32000, Israel
[ameer@tx, ran@ee, kolodny@ee].technion.ac.il, friedman@ece.rochester.edu

ABSTRACT
Clock skew variations adversely affect timing margins, limiting performance, reducing yield, and may also lead to functional faults. Non-tree clock distribution networks, such as meshes and crosslinks, are employed to reduce skew and also to mitigate skew variations. However, these networks incur an increase in dissipated power while consuming significant metal resources. Several methods have been proposed to trade off power and wires to reduce skew. In this paper, an efficient algorithm is presented to reduce skew variations rather than skew, and prioritize the algorithm for critical timing paths, since these paths are more sensitive to skew variations. The algorithm has been implemented for a standard 65 nm cell library using standard EDA tools, and has been tested on several benchmark circuits. As compared to other methods, experimental results show a 37% average reduction in metal consumption and 39% average reduction in power dissipation, while insignificantly increasing the maximum skew.

Categories and Subject Descriptors
B.7.2 [**INTEGRATED CIRCUITS**]: Design Aids

General Terms
Algorithms, Design, Performance

Keywords
Clock distribution, non-tree clock networks, clock mesh synthesis, clock skew, process variations, power, VLSI CAD, physical design

1. INTRODUCTION
Non-tree clock distribution topologies (*e.g.*, clock meshes) exhibit useful characteristics due to multi-path signal propagation created by routing redundancies [1–11]. These non-tree clock distribution networks are exploited to distribute the global clock signal over an integrated circuit, and exhibit high immunity to process, voltage, and temperature (PVT) variations, while tolerating non-uniform switching and an unbalanced distribution of the clocked elements. These networks achieve low and deterministic skew, low skew variations, and low jitter. Clock meshes also overcome late design changes while satisfying tight time-to-market deadlines [1]. Clock meshes constitute an effective alternative for distributing global clock signals, and are used in high performance microprocessors [1] such as the Power4 [2], Digital Alpha [3], Intel Pentium 4 [4], and Xeon [5].

Nevertheless, non-tree clock distribution networks suffer certain drawbacks. These networks are composed of a large number of mesh nodes and unbalanced loads, making these networks difficult to analyze, optimize, and automate [6],[12],[13]. Routing redundancies require significant resources as compared to optimized tree-based clock distribution networks where point-to-point routing is used [7]. Meshes dissipate higher power [12] due to the large capacitance incurred by the additional metal wires and drivers. Furthermore, clock gating is impractical in most mesh structures. Due to delay differences in the drivers, short-circuit current loops are generated across the redundant mesh paths [12]. Increasing process variations [14],[15] dissipate more power, since a more tolerant mesh structure dissipates higher power due to greater use of metal and driver oversizing [15]. Several proposals for optimizing non-tree distribution networks have been presented, employing either customized meshes [1–7] or automating the process of adding crosslinks to the clock tree to enhance tolerance and lower power [8],[9]. Yet other papers propose removing some edges from a mesh to reduce power while minimally increasing the skew [10],[11].

While most of these papers focus on skew variations, the approach proposed in this paper manages skew tolerance based on the criticality of the timing margins. The clocks driving a critical logic path are required to be more tolerant to skew variations to reduce the effect of skew on timing margins and cycle time. Those clocks that drive a non-critical logic path however must satisfy certain skew variations without affecting the cycle time [16]. By relaxing skew variation requirements in the non-critical paths, power savings can be achieved. The proposed method employs graph-theoretic and geometric algorithms with quasi-linear run time. Using static timing analysis (STA), a physical floorplan, and process information, a non-uniform clock mesh which tolerates clock skew based on timing path criticality is generated. The proposed flow has been successfully tested on several testbench circuits.

The rest of the paper is organized as follows. Non-tree clock distribution networks, skew constraints, constraint graphs, and clock skew uncertainty are reviewed in section 2. The motivation behind this work and a review of previous work on clock mesh synthesis and optimization are discussed in section 3. The timing–driven variation–aware clock mesh synthesis problem and the proposed solution are presented along with a run time example in Section 4. The experimental method and results are described in section 5. Finally, this paper is concluded in section 6 with suggestions for future research directions.

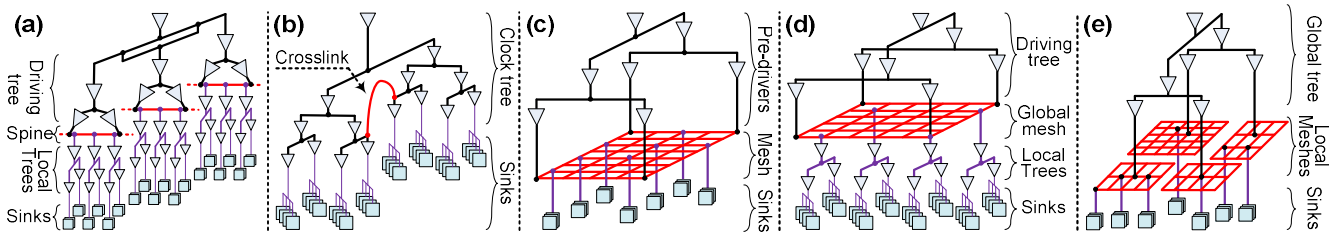

Figure 1: Non-tree and mesh clock architectures: (a) Pentium 4 spine, (b) tree with crosslinks, (c) leaf level global mesh, (d) global mesh with local trees (MLT), and (e) global tree with local meshes (TLM)

2. CLOCK MESHES AND SKEW

In this section, essential preliminaries are outlined. Specifically, in section 2.1, non-tree clock topologies, and in particular, clock meshes are described. In section 2.2, the concept of clock skew, skew uncertainty modeling and notation, and constraint graphs for representing synchronous circuits are reviewed.

2.1 Clock Meshes

Clock tree topologies provide a single path for each sink. For distant sinks, these paths are largely separate from each other. Each separate path suffers from delay uncertainty, resulting in skew uncertainty between two sinks. One approach to reduce variations is simply connecting nodes with a crosslink, hence, the connected nodes will have more than one path from the clock source, mutually compensating each path. This non-tree approach has been manually applied to the Pentium 4 microprocessor [4], where spines connect multi-clock nodes, as illustrated in Fig. 1(a). Automation of this method has been evaluated in several papers [8].

Since spines or crosslinks connect pairs of nodes and do not cover an entire floorplan (see Fig. 1(b)), metal grids driven by a top level clock driver which span several regions have been introduced. A mesh is a grid of horizontal and vertical metal wire segments, composed of interconnected mesh nodes. Typical mesh topologies consist of three parts: the mesh itself (usually uniform), an upper driving tree, and local interconnects connecting the clock sinks to the mesh, as shown in Fig. 1(c). A wide variety of mesh structures has been proposed. Non-uniform meshes [5],[10],[11] have been developed to save wire resources and power. Design automation and optimization of metal and power versus tolerance tradeoffs are discussed in [10],[11]. Mesh architectures differ in the locality of the mesh. A global mesh with local trees (MLT) [6] is a mesh fed from a global clock source with local trees distributing the clock to local regions (Fig. 1(d)), while a global tree with local meshes (TLM) [6] is a clock tree fed from a global clock source with a local mesh at each leaf (Fig. 1(e)). Other hybrid mesh-style structures are also possible.

2.2 Skew constraints and uncertainty

Synchronous circuits comprise data paths, where each data path consists of combinational logic located between two registers. A clock network connects the clock source to a collection of clock sinks $S=\{s_1,s_2,\ldots,s_n\}$. Two registers are *sequentially-adjacent* if the registers are connected with a combinational data path [1], as illustrated in Fig. 2. The maximum permissible delay for a local data path bounded by two sequentially-adjacent registers is $P^{i,j}_{Delay,max}=T_{clock}-t_{setup}-skew^{i,j}_{max}$ where T_{clock} denotes the clock period, t_{setup} is the setup time of the bounding registers, and $skew^{i,j}_{max}$ denotes the maximum clock skew between two bounding registers [1]. The maximum clock skew $skew^{i,j}_{max}$ is the difference in the clock arrival time between two *sequentially-adjacent* registers. If d_i and dj are the delays (maximum or minimum) of the clock signals arriving at the registers i and j, respectively, the skew between two adjacent registers is $skew^{i,j}_{max}=max_{i,j}(|d_i-dj|)$. The clock skew is therefore bounded by the following *maximum skew constraint*, $skew^{i,j}\le T_{clock}-t_{setup}-P^{i,j}_{Delay,max}$.

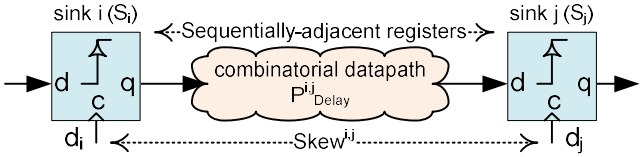

Figure 2: *Sequentially-adjacent* registers s_i and s_j with $skew^{i,j}$ clock skew, bounding a combinational data path with propagation delay $P^{i,j}_{Delay}$.

Clock skew uncertainty: As technology scales, the effect of process variations on clock skew is aggravated [14],[15]. Clock skew can be modeled as composed of both deterministic and probabilistic elements [13],[18].

In this work, the following notation is employed. For two sequentially-adjacent registers i and j, the deterministic or nominal skew components is $skew^{i,j}_{nom}$, and the maximum skew variation of the probabilistic component is $\delta^{i,j}_{max}$. The mean of the skew between two sequentially-adjacent registers i and j is denoted by $\mu^{i,j}_{skew}$ and the standard deviation is denoted by $\sigma^{i,j}_{skew}$. A possible design target may require that, for instance, the maximum skew will be limited by $skew^{i,j}_{max}=\mu^{i,j}_{skew}+3\cdot\sigma^{i,j}_{skew}$.

Constraint graph: Synchronous circuits are represented as a directed multi-graph G_C [1],[16],[17]. Each clock sink is represented by a vertex $v_i\in G_C{}^V$, so that $G_C{}^V = S$. Each local data path located between two sequentially-adjacent clock sinks i and j is represented by a weighted edge $e_{i,j}\in G_C{}^E$ connecting the two vertices v_i and v_j. The graph edges are therefore $G_C{}^E=\{e_{i,j}=v_i\sim v_j \mid P^{i,j}_{Delay}<\infty,\ v_i,v_j\in G_C{}^V\}$ (see Fig. 3). The edges can be weighted by any corresponding combinational data path property, such as delay, margin, and skew. Besides the edge weights, attributes can be attached to either edges or vertices. For vertices, any corresponding sink attribute can be used, such as the sink capacitance, location, clock delay, and data arrival time.

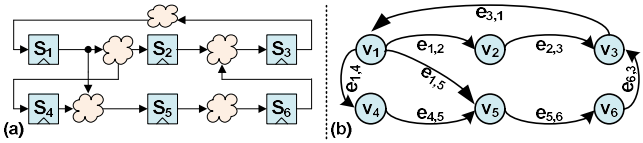

Figure 3: Mapping of constraint graph: (a) synchronous circuit and (b) corresponding constraint graph.

3. CLOCK MESH SYNTHESIS: MOTIVATION AND RELATED WORK

Motivation for using timing information as a criterion for clock mesh syntheses is discussed in section 3.1. A review of previous clock mesh synthesis methods is presented in section 3.2.

3.1 Motivation

Aggressive process scaling increases the portion of clock skew as compared to the cycle time, reducing timing margins [14],[15]. Some approaches have been proposed to minimize clock skew, but these methods usually incur an increase in power consumption. Other methods exploit useful skew by scheduling clock skew to increase the maximum frequency [17]. These methods, however, suffer from increased clock skew uncertainty with process scaling. This issue limits circuit performance since timing margins are reduced [14],[15].

In order to overcome uncertainty in the clock arrival time caused by within-die variations, timing margins are provided. While poor margins reduce yield, extreme worst case margins can produce overdesigned circuits with increased power dissipation and resource consumption.

In particular, the critical paths within a digital circuit are sensitive to skew variations. Hence, skew variations should be minimized, particularly in those clock paths that drive the critical paths. This method, however, is usually achieved at the expense of higher power dissipation [1]. This paper focuses on differential treatment of the clock and selective management of the skew. The more critical a path, the more sensitive the path is to skew variations and the greater the effort to reduce these skew variations [16]. Paths that are non-critical are assigned a lower effort to reduce skew variations. Path criticality prioritization is intended to save power since skew variations are not minimized on those paths that do not affect circuit speed. This approach is in contrast to skew optimization methods that aim to reduce maximum or nominal skew over an entire circuit.

3.2 Related work

The effect of increasing process variations is particularly pronounced in clock distribution networks, since skew variations strongly influence system performance and require careful treatment of minimum delays [14],[15]. Non-tree clock meshes, although useful in mitigating process variations, are difficult to analyze and automate due to the complex structure; most mesh clock networks are manually designed in high performance applications such as microprocessors [1–7]. Several approaches automate the clock mesh design process. Mesh sizing and, in particular, segment wire width sizing using network flow algorithms have been used to optimize nominal skew rather than skew variations [19]. Other methods start from a clock tree and incrementally add crosslinks among the tree nodes or leaves. Crosslinks are added between those nodes exhibiting high variation. The objective is to add the fewest number of crosslinks that can reduce the maximum or overall variations [8],[9]. Other approaches start with a fully uniform mesh, identify and remove redundant segments whose effect on variations is minimal by applying network theory techniques, thereby trading off variations with wire length. A set-cover problem is solved to obtain mesh pre-driver minimum buffers [10],[11]. The initial uniform grid is designed to ensure that metal redundancies within the grid satisfy target skew requirements [10].

4. TIMING–DRIVEN VARIATION–AWARE CLOCK MESH SYNTHESIS

The timing-driven variation-aware nonunifrom clock mesh synthesis problem is presented in this section. The problem formulation and solution approach are presented in sections 4.1 and 4.2, respectively.

4.1 Problem statement

The problem of managing skew variations can be formulated using the notations defined in Section 2.2:

Inputs: Given a circuit connectivity and static timing analysis, including (1) a set of clock sinks $S=\{s_1,s_2,\ldots,s_n\}$, (2) maximum skew constraints between each set of sequentially-adjacent registers s_i and s_j, namely, the maximum permissible skew $skew^{i,j}$, and (3) the relative tolerance parameter ξ, a user defined parameter denoting the upper bound of the maximum skew variation ratio over all maximum skew constraints allowed for all data paths,

$$(\forall e_{i,j} \in G_C^E) \;\; \xi \geq \delta^{i,j}{}_{max} / skew^{i,j}. \qquad (1)$$

Problem Formulation: Construct a clock mesh with reduced wire length and power consumption that limits the fraction of the maximum skew variation over all maximum skew constraints by ξ for every combinational data path, as expressed by equation (1).

The mesh density is the number of nodes connecting wire segments within the mesh. A uniform mesh consists of m horizontal segments and n vertical segment requiring $n \times m$ nodes.

Higher levels of ξ lead to further reductions in clock skew variations at the expense of a denser mesh, longer wire length, and higher power consumption. ξ may be tuned by considering the tradeoff among power dissipation, metal consumption, and design robustness.

4.2 Mesh construction algorithm

The algorithm places multiple clock meshes over certain rectangular regions. The regions may partly overlap. The meshes may be of different densities. Each region covered by a mesh is associated with a specific maximum skew constraint, which determines the density of the corresponding mesh. The algorithm comprises four phases.

Phase I: A constraint graph, as defined in Section 2.3 above, is derived from circuit connectivity information. See the example shown in Fig. 4. The vertices represent clock sinks and the edges represent data paths between vertices. The edge weight $w(e_{i,j})$ represents the maximum skew constraint of the local data path represented by the edge $e_{i,j}$, $w(e_{i,j})=skew^{i,j}$. Each vertex can be assigned multiple attributes, such as the capacitance of the corresponding sink $C(v_i)=Capacitance(s_i)$ and the geometric location of the clock sinks of the vertex.

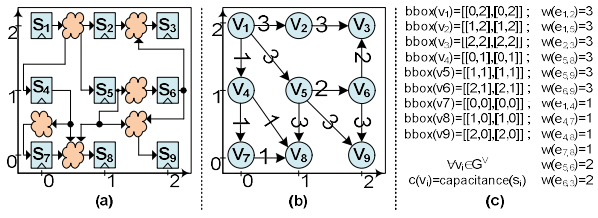

Figure 4: Phase I, constraint graph construction: (a) synchronous circuit floorplan and connectivity with placed registers, (b) corresponding constraint graph; edge weights are the maximum allowed skew, and (c) vertex attributes and edge weights are the initial values

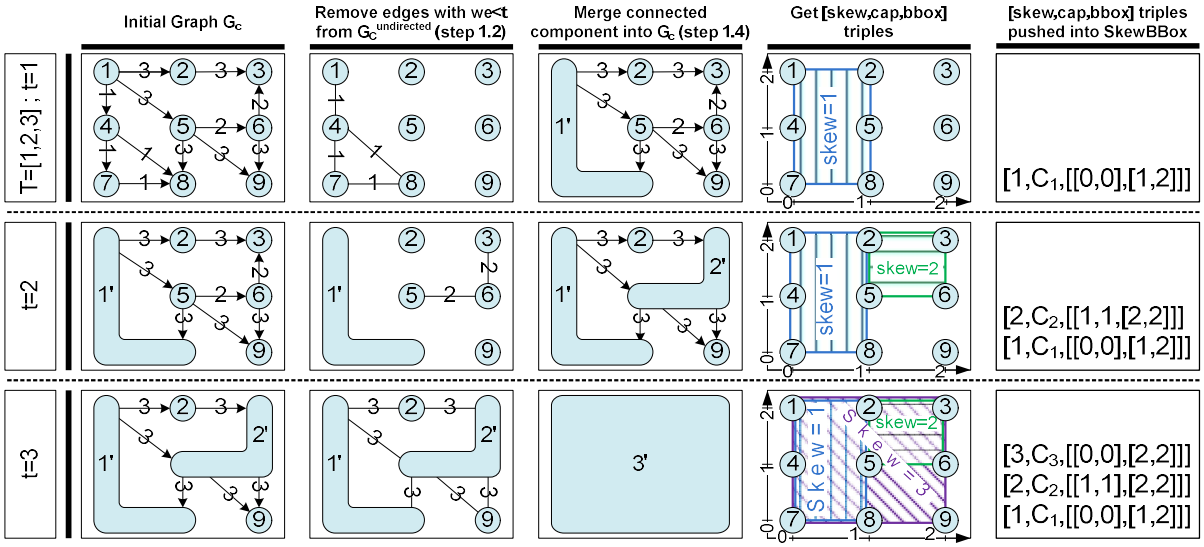

Figure 6: Phase II execution example. Rows are iterations and columns are steps of the algorithm

As the algorithm progresses, some vertices are merged, representing a geometric rectangular region containing multiple clock sinks. As the vertices are merged, the inner connectivity between the constituent sinks is ignored, and only the inter-vertex connectivity is preserved. The attributes of a vertex represent the properties of all sinks included in the vertex: the capacitance is the sum of all sink capacitances and the geometric location is replaced by a rectangular bounding box covering the physical location of all of the corresponding clock sinks.

Misplaced registers, *e.g.*, sequentially-adjacent registers bounding a critical timing path and placed diagonally apart, may cause undesired results. These cases are reported at an early stage, suggesting replacing interfering registers to avoid unnecessary design loops.

Phase II: The rectangular regions satisfying certain skew requirements are identified. The algorithm iterates over an increasing threshold level. A vector T contains all pre-determined threshold values and defines basic time steps for mesh construction. At each iteration, each connected vertex is interconnected by edges with a weight below the current threshold and merged into a new vertex. A geometric bounding box covering all clock sinks within the same vertex is identified. The algorithm is described in Fig. 5. A threshold $t \in T$ is selected, edges with weight less than the threshold are eliminated (step 1.2), the corresponding vertices are identified as a connected component (step 1.3), and these vertices are merged into one larger vertex (step 1.4.1). The skew constraint of this new vertex is the tightest skew constraint among all edges inside the corresponding connected component (step 1.4.2). The algorithm terminates when no threshold values remain in T, or only one vertex remains. A run time example of Phase II is shown in Fig. 6. The merge operation is performed by the subroutine shown in Fig. 7, as follows: The inner edges inside a connected component are removed and externally connected edges are connected to the new merged vertex (step 2). The attributes of all vertices inside the connected component are merged into the attributes of the corresponding new vertex. The capacitance is the sum of all inner capacitances (step 3) and the bounding box bounds all inner sinks or inner bounding boxes (step 4). The merge operation is illustrated in Fig. 8. Note that this phase produces a set of possibly overlapping rectangular regions.

Generate *[skew,capacitance,bounding box]* triplets	
Inputs:	G_C: Constraints graph, T: Thresholds vector
Output:	*skewBbox* stack, contains $[skew_{ucc}, cap_{ucc}, bbox_{ucc}]$ triplets, in ascending order by skew

1. foreach $t \in T$
  1.1. $G_C^{undirected} = getUndirected(G_C)$
  1.2. foreach $e \in G_C^E$
    1.2.1. if $w_e < t$ $G_C^{undirected} = G_C^{undirected} / e$
  1.3. $UCC = getConnectedComponents(G_C^{undirected})$
  1.4. foreach $ucc \in UCC$
    1.4.1. $v_{merge} = mergeVertices(G_C, ucc)$
    1.4.2. $skew_{ucc} = min(w_e | e = v_i \sim v_j,\ v_i, v_j \in ucc)$
    1.4.3. $bbox_{ucc} = bbox(v_{merge})$
    1.4.4. $cap_{ucc} = cap(v_{merge})$
    1.4.5. $push(skewBbox, skew_{ucc}, cap_{ucc}, bbox_{ucc})$

Figure 5: Phase II algorithm for generating *[skew, cap., bounding box]* triplets (*mergeVertices()* is shown in Fig. 7)

Merge vertices procedure:	
Inputs:	G: Graph, V: Group of vertices to be merged
Output:	V_{merge}: Merged vertex with merged attributes

MergeVertices(G,V)
1. create new vertex v_{merge}
2. foreach $v \in V$
  2.1. foreach $e = v \sim v'$
    2.1.1. if $v' \notin V$ $e = v' \sim v_{merge}$
  2.2. remove v
3. $C(v_{merge}) = \sum_{v \in V} C(v)$
4. $Bbox(v_{merge}) = [min(x_0), min(y_0), max(x_1), max(y_1)]$ |
$(x_0, y_0, x_1, y_1) = bbox(v),\ v \in V$
5. return v_{merge}

Figure 7: Vertex merging algorithm (part of Phase II)

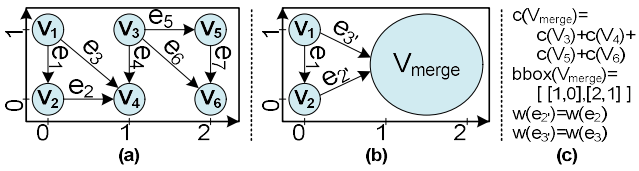

Figure 8: Merge example: (a) constraint graph, vertices are placed at the same place as the corresponding registers, (b) constraint graph after merge, and (c) values of attributes after merge

Generate *[skew,capacitance,polygon]* triplets:	
Inputs:	*[skew,capacitance,bbox]* triplets from phase II
Output:	*skewPolygon* stack, containing *[skew,capacitance,polygon]* triplets
1. covered=Ø	
2. while skewBbox≠Ø	
2.1. [skew,cap.,bbox]= pop(reversed((skewBbox))	
2.2. polygon=covered^c∩bbox	
2.3. covered=covered∪bbox	
2.4. *push(skewPolygon, [skew,capacitance,polygon])*	

Figure 9: Generating *[skew,cap.,polygon]* triplets (part of Phase III)

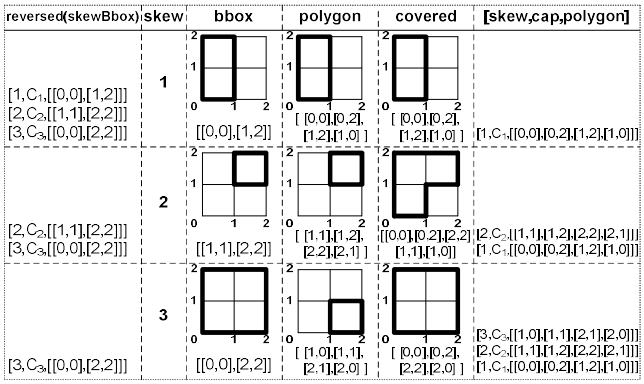

Figure 10: Phase III execution example. Rows are iterations and columns are steps of the algorithm

The time complexity of Phase II is $O(|S|)$. Extracting and merging all connected components requires $O(|G_C^V|+|G_C^E|)$. The overall run time of phase II is therefore $O(|T|\cdot(|G_C^V|+|G_C^E|))$. Since $|T|$ is constant and the number of edges is of the same order as the number of vertices $|G_C^E|=O|G_C^V|$, the time complexity of phase II is $O|G_C^V|=O(|S|)$.

Phase III: Partly overlapping rectangular skew regions are merged into polygons, and skew levels are assigned to the polygons. When two skew regions overlap, the tighter skew constraint prevails and is inherited by the resulting polygon, as illustrated in Fig. 9. The input *[skew, capacitance, bounding box]* triplets are sorted in ascending order of skew. Iteratively, a triplet with a tighter skew constraint is removed from the input stack (step 2.1). The circuit floorplan is filled with non-overlapping skew regions (steps 2.2 and 2.3) and polygon shaped skew regions are constructed from overlapping regions. A run time example of Phase III is shown in Fig. 10.

Polygon union and intersection operations can be performed in $O(n\cdot log(n))$ steps using a segmented tree data structure [20], where n is the total number of polygon segments. This complexity is the same order as the number of vertices $n=O(|G_C^V|)=O(|S|)$, and the run-time of this phase is therefore quasi-linear, $O(|S|\cdot log(|S|))$.

Final Phase IV: A clock mesh is designed for each of the non-overlapping skew polygons. Each mesh should satisfy the skew requirement of the polygon. If the maximum allowed skew for a specific region is $skew_{max}$, the maximum allowed variation is $skew_{max} < \xi \cdot skew_{max}$. The skew variation is inversely proportional to the mesh density. The density of each mesh is therefore tuned to match the required skew variation. Skew as a function of mesh density has also been discussed in [10]. Optimized pre-drivers are placed by solving a set-covers problem [10],[11].

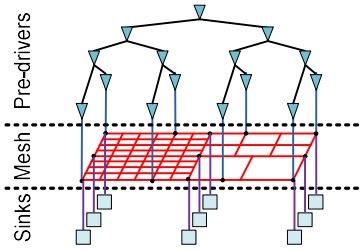

Figure 11: Example of the proposed mesh

The overall run time for the entire algorithm is quasi-linear in the number of clock sinks: $O(|S|\cdot log(|S|))$. An example output of the algorithm is illustrated in Fig. 11.

5. EXPERIMENTAL RESULTS

In order to verify the capability of the proposed method to reduce power and wire length consumed by the mesh architecture, and compare these capabilities to previous methods, several experiments have been conducted. These experiments are described and the results are discussed in sections 5.1 and 5.2, respectively.

5.1 Flow and design environment

The proposed algorithm has been implemented in Perl and TCL. Experiments have been performed on several circuits from the ISCAS89 sequential benchmark suite. These benchmark circuits have been designed using the Virage Logic standard cell logic library with a 65 nm process operating at a 1 GHz frequency. RTL representations of the benchmark circuits have been synthesized into a netlist using Synopsys Design Compiler Ultra (DC Ultra) and placed and routed by Cadence SoC Encounter™ RTL-to-GDSII System. A TCL hook procedure is used to construct the constraint graphs, which are imported into a XML database. The proposed algorithm generates mesh and pre-driver locations. The Cadence SoC Encounter constructs the final physical layout. Results are analyzed using Cadence Virtuoso UltraSim Full-Chip Simulator, a transistor-level FastSPICE circuit simulator.

5.2 Results

The results of applying the proposed algorithm to several benchmark circuits are listed in Table 1. The first two columns list the benchmark name and number of registers. The following three columns list, respectively, the total wire length, power consumption, and maximum skew. The experimental parameter ξ is varied around a typical value. The experiment evaluates the

Table 1: Results of the proposed algorithm as compared with other approaches [10], [11]

Testcase	#Sinks	Wire Length (um)					Power (mw)					Maximum skew (ps)				
		ξ = 1.1	ξ = 1.0	ξ = 0.9	[10]	[11]	ξ = 1.1	ξ = 1.0	ξ = 0.9	[10]	[11]	ξ = 1.1	ξ = 1.0	ξ = 0.9	[10]	[11]
s9234	135	12490	13376	13857	33610	27177	5.03	5.27	6.8	8	6.7	113.7	93.7	82.5	163.9	124.1
s5378	165	20149	20839	22165	31009	24911	4.57	4.84	4.92	6.7	6.72	128.3	106.2	94.1	204.16	128.06
s13209	500	51220	51443	52834	82884	109538	11.8	12.59	12.91	20.6	23.8	132.1	111.4	91.8	62.33	95.81
s15850	566	44535	45628	45735	84055	100778	13.32	13.93	14.72	22	23.8	126	102.1	85.5	94	106.64
s38584	1426	164224	166274	165372	256567	262528	42.36	41.18	43.55	65.2	60.9	150.2	131.1	104.6	182.28	130.7
s35932	1728	237513	239342	241680	349432	321293	43.14	44.25	41.83	73.5	74.3	153.6	124.6	107.2	108.5	134.65
Average	753.3	88355.2	89483.7	90273.8	139593	141038	20	20.3	20.8	32.7	32.7	134	111.5	94.3	135.9	120

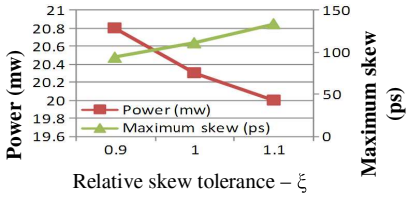

Figure 12: Power and maximum skew vs. relative skew tolerance parameter ξ

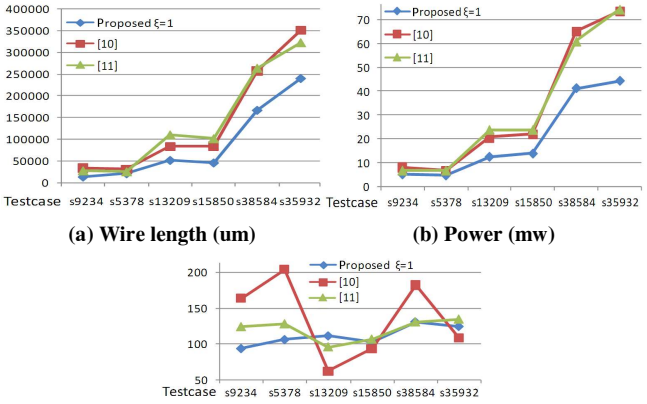

Figure 13: Proposed method with ξ = 1 vs. [10] and [11]: (a) wire length, (b) power, and (c) maximum skew

relationship among the metal resources, power consumption, and relative skew tolerance parameter ξ. The maximum skew is compared with other methods, and, as expected, a larger ξ increases wire length and power consumption but reduces maximum skew (see Fig. 12). Comparing these results to the methods proposed in [10],[11], a typical value of $\xi = 1$ improves the wire length and power dissipation with an insignificant increase in maximum skew, as depicted in Fig. 13. As compared to [10] and [11], the proposed method achieves a 37% average reduction in metal consumption and a 39% average reduction in power dissipation. These results demonstrate that managing skew tolerance by wisely prioritizing critical paths saves significant metal resources and dissipates less power as compared to traditional methods.

6. CONCLUSIONS

An efficient graph-theoretic and geometric quasi-linear algorithm for managing clock skew tolerance is presented in this paper. Skew variations are managed while considering the criticality of the timing of each data path. An algorithm and flow for planning and synthesizing non-uniform clock meshes are integrated with current CAD tools and demonstrated on a 65 nm CMOS process and cell library. Experimental results on a set of benchmark circuits exhibit a 37% average decrease in metal wire length and a 39% average decrease in power dissipation with an insignificant increase in maximum skew as compared to existing methods.

Future improvements of the algorithm should be considered. Other parameters, *e.g.*, metal width and layer, could be integrated into the optimization process. Since constraint graph extraction is computationally expensive, registers at the same local region could be clustered into one node before extraction. Rather than a global mesh that directly drives the clock sinks, a hybrid clock mesh topology may be employed, *e.g.*, a global mesh with local trees (MLT) [6]. The algorithm presented here can be adapted to automate the crosslink insertion process [8]. Rather than inserting crosslinks to reduce maximum variations, the crosslinks could be inserted according to the criticality of the individual data paths. The algorithm presented here targets a mesh for zero-skew; useful skew may also be considered.

7. ACKNOWLEDGMENTS

The authors would like to thank the VLSI Systems Research Center at the Technion for providing CAD tools, and Virage Logic Corp. for providing technology process and cell library. The research is supported in part by the Technion ACRC and by the ALPHA research consortium.

8. REFERENCES

[1] E. G. Friedman, "Clock Distribution Networks in Synchronous Digital Integrated Circuits," *Proceedings of the IEEE*, Vol. 89, No. 5, pp. 665-692, May 2001.

[2] P. J. Restle *et al.*, "The Clock Distribution of the Power4 Microprocessor," *Proceedings of the IEEE International Solid-State Circuits Conference*, pp. 1.144-1.145, February 2002.

[3] T. Xanthopoulos *et al.*, "The Design and Analysis of the Clock Distribution Network for a 1.2 GHz Alpha Microprocessor," *Proceedings of the IEEE International Solid-State Circuits Conference*, pp. 402-403, February 2001.

[4] N. A. Kurd *et al.*, "A Multigigahertz Clocking Scheme for the Pentium® 4 Microprocessor," *IEEE Journal of Solid-State Circuits*, Vol. 36, No. 11, pp. 1647-1653, November 2001.

[5] S. Tam *et al.*, "Clock Generation and Distribution of a Dual-Core Xeon Processor with 16MB L3 Cache," *Proceedings of the IEEE International Solid-State Circuits Conference*, pp. 1512-1521, February 2006.

[6] C. Yeh *et al.*, "Clock Distribution Architectures: a Comparative Study," *Proceedings of the IEEE International Symposium on Quality Electronic Design*, pp. 85-91, March 2006.

[7] P. J. Restle and A. Deutsch, "Designing the Best Clock Distribution Network," *Proceedings of the IEEE Symposium on VLSI Circuits*, pp. 2-5, June 1998.

[8] A. Rajaram, J. Hu, and R. Mahapatra, "Reducing Clock Skew Variability Via Crosslinks," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 25, No. 6, pp.1176-1182, June 2006.

[9] I. Vaisband, R. Ginosar, A. Kolodny, and E. G. Friedman, "Power Efficient Tree-Based Crosslinks for Skew Reduction," *Proceedings of the ACM/IEEE Great Lakes Symposium on VLSI*, pp. 285-290, May 2009.

[10] A. Rajaram and D. Z. Pan, "MeshWorks: an Efficient Framework for Planning, Synthesis and Optimization of Clock Mesh Networks," *Proceedings of the IEEE Asia and South Pacific Design Automation Conference*, pp. 250-257, January 2008.

[11] G. Venkataraman, Z. Feng, J. Hu, and P. Li "Combinational Algorithms for Fast Clock Mesh Optimization," *Proceedings of the IEEE/ACM International Conference on Computer-Aided Design*, pp. 563-567, November 2006.

[12] A. L. Sobczyk, A. W. Luczyk, and W. A. Pleskacz, "Power Dissipation in Basic Global Clock Distribution Networks," *Proceedings of the IEEE Design and Diagnostics of Electronic Circuits and Systems*, pp. 1-4, April 2007.

[13] G. Tosik, L. M. S. Gallego, and Z. Lisik, "Different Approaches for Clock Skew Analysis in Present and Future Synchronous IC's," *Proceedings of the International Conference on "Computer as a Tool,"* pp. 1227-1232, September 2007.

[14] V. Mehrotra and D. Boning, "Technology Scaling Impact of Variation on Clock Skew and Interconnect Delay," *Proceedings of the IEEE International Interconnect Technology Conference*, pp. 122-124, June 2001.

[15] S. Abe, M. Hashimoto, and T. Onoye, "Clock Skew Evaluation Considering Manufacturing Variability in Mesh-Style Clock Distribution," *Proceedings of the IEEE International Symposium on Quality Electronic Design*, pp. 520-525, March 2008.

[16] D. Velenis, M. C. Papaefthymiou, and E. G. Friedman, "Reduced Delay Uncertainty in High Performance Clock Distribution Networks," *Proceedings of the IEEE Design Automation and Test in Europe Conference*, pp. 68-73, March 2003.

[17] I. S. Kourtev, B. Taskin, and E. G. Friedman, *Timing Optimization Through Clock Skew Scheduling*, Second Edition, Springer Science+Business Media, 2009.

[18] S. D. Kugelmass and K. Steiglitz, "A Probabilistic Model for Clock Skew," *Proceedings of the IEEE International Conference on Systolic Arrays*, pp. 545-554, May 1988.

[19] M. P. Desai, R. Cvijetic, and J. Jensen, "Sizing of Clock Distribution Networks for High Performance CPU Chips," *Proceedings of the IEEE/ACM Annual Design Automation Conference*, pp. 389-394, June 1996.

[20] F. P. Preparata and M. I. Shamos, *Computational Geometry: an Introduction*, Springer-Verlag, 1985.