Zero-Skew Clock Tree Construction by Simultaneous Routing, Wire Sizing and Buffer Insertion*†

I-Min Liu[1] Tan-Li Chou[2] Adnan Aziz[1] D.F. Wong[3]
[1]Electrical and Computer Engineering, The University of Texas at Austin, Austin, Texas 78712
[2]Strategic CAD Labs., Design Technology, Intel Corporation, Hillsboro, Oregon 97124
[3]Computer Sciences, The University of Texas at Austin, Austin, Texas 78712

ABSTRACT

We propose an integrated clock tree construction algorithm which performs simultaneous routing, wire sizing and buffer insertion. In existing approaches, wire sizing and clock buffer insertion are typically applied sequentially after a clock tree is generated and routed, i.e., they are done as post-processing steps. None of the known methods can perform clock routing while simultaneously considering wire sizing and buffer insertion. We introduce wire widths and levels of buffers inserted as variables in forming merging segments in the proposed *Integrated Deferred-Merge Embedding* (IDME) algorithm. As a result, more zero-skew merging locations are made possible and the clock trees generated are zero-skew by construction. Our experiments show that by taking the advantage offered by wire sizing, we are able to minimize phase delay as well as to reduce wire length and use less buffers.

1. INTRODUCTION

In high performance synchronous VLSI design, system performance is limited by circuit delays and clock skews. Clock skew is defined as the maximum difference in arrival times of different receivers (from a clock source to these receivers). Without careful design, clock skews can cause lower clock frequency (*zero clocking*) as well as race conditions that result in failure regardless of frequency (*double clocking*). Therefore, it is imperative to design a clock distribution network with minimum skew to simplify logic synthesis and timing verification. Although non-zero clock skew can be exploited to improve timing margins [1; 2], in practice, it involves the interaction among logic synthesis, timing convergence, and clock routing. It remains to be an active research area. We will not address the schedule synthesis that allocates and utilizes useful clock skew. Instead, we will focus on minimizing clock skew in clock routing.

Keeping clock skew manageable is becoming more difficult in deep submicron environment due to tighter budget (faster clock signal), more process variation and increasing interconnect delays. Skew is caused by many factors, e.g., load mismatch, process variation, and power supply noise. Power supply noise can be dealt with careful power grid design. While buffer insertion improves slew rate (slope) of clock signals and power, it is subject to process variation [3]. By having the same number of buffer stages on any root to leaf path [4] (levelized buffer insertion), the impact of process variation can be reduced. That is, the phase delay (the maximum delay from a source to receivers) change due to die-to-die process variation will have the same effect on all the receivers.

1.1 Previous Work

There have been a number of approaches to solving the load mismatch problem. Zero-skew routing was first proposed in [5]. Wirelength reduction was achieved by Deferred-Merge Embedding (DME) algorithm [6]. Further wirelength reduction was made possible by nearest neighbor selection or clustering-based algorithm [7]. DME has also been extended to planar routing [8; 9]. Lately, researchers discovered how to further minimize wirelength by using bounded skew [10; 11; 12; 13]. Instead of merging segments, merging regions are formed providing more possible locations during the bottom-up construction of clock routing. Boundary segments or sampled segments of the merging regions are used to form merging regions of the next-level of hierarchy. However, none of the above approaches consider phase delay.

In order to maintain the quality of clock signals and to reduce power consumption, in high performance design, phase delay should be controlled and minimized. Simulated annealing based zero-skew clock net construction algorithm was proposed in [14] to minimize phase delay. Lower and upper bounded delay routing was considered using linear programming [15]. Neither considered sizing of wire width, which can reduce delay dramatically. The author of [7] minimized delay by choosing a wire width for each merging segment. But different merging segments due to different wire widths, which may reduce wirelength and phase delay, were not explored. Various wire width sizing techniques have been proposed [16; 17; 18; 19] as a post-processing step af-

*The work of Adnan Aziz was partially supported by an NSF Career Award, by the Texas Advanced Technology Program under Grant No. 0036580235, and by a gift from IBM Corporation.
†The work of D.F. Wong was partially supported by the Texas Advanced Research Program under Grant No. 003658288 and by a grant from the Intel Corporation.

ISPD 2000, San Diego, CA.

sertion are applied to a given clock routing. As a result, the quality of the solutions depends on the initial routing and can not guarantee zero-skew.

There are three known methods in integrating buffer insertion and clock routing [26, 27, 28]. In the first method [26], buffer placement was integrated into DME [26] to avoid buffer overlapping. However, wire sizing is not considered and buffer insertion did not directly take phase delay into account. In the second method [29, 27], simulated annealing method was adopted from [14] to generate a clock topology. After a topology was generated, routing, wire sizing [29], and buffer insertion [27] were applied *sequentially* to evaluate the cost function. Therefore, wire sizing and buffer insertion were in effect added as post processing steps of routing. In the third method [28], the bottom-up phase of DME has been extended to insert buffers at different locations while preserving zero-skew. It has been shown that power, slew rate (rise and fall time), and wirelength can be improved due to the added freedom [28]. This motivates us to explore the interaction among routing, wire sizing, and buffer insertion simultaneously during the merging process.

1.2 Our Work

We propose an integrated clock tree construction algorithm which performs simultaneous routing, wire sizing and buffer insertion. In existing approaches, wire sizing and clock buffer insertion are typically applied sequentially after a clock tree is generated and routed, i.e., they are done as post-processing steps. None of the known methods can perform clock routing while simultaneously considering wire sizing and buffer insertion.

We introduce wire widths and levels of buffers inserted as variables in forming merging segments in the proposed *integrated deferred-merge embedding* (IDME) algorithm. As a result, more zero-skew merging locations are made possible and the clock trees generated are of zero-skew by construction. We generalize the concept of merging segments to that of levelized Merging Segment Set (MSS). In order to keep the size of levelized MSS tractable, we also propose pruning and sampling techniques to reduce the size.

The remainder of this paper is organized as follows. Section 2 reviews basic aspects of DME. Section 3 describes how wire sizing can be considered in DME. Section 4 gives details of how levelized buffer insertion scheme is integrated along with wire sizing. Experimental results are presented in Section 5 while the findings and conclusions are summarized in Section 6.

2 Review of DME Algorithm

Below is a brief review of the DME algorithm; readers are referred to [6] for details. Given a set of clock pins $S = \{s_1, \cdots, s_n\}$ and a connection topology G, the DME algorithm finds physical locations for nodes in G to find a zero-skew tree (ZST) T. A connection topology is a rooted binary tree G, which has n leaves corresponding

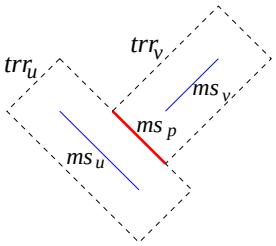

Figure 1. Construction of a merge segment.

to the n clock pins. The internal nodes are the nodes that are not clock pins. Let $d(u, v)$ be the delay from node u and node v. A ZST is a tree such that for any non-clock pins $max_{1 \le i,j \le n} |d(v - s_i) - d(v - s_j)|$ is zero. The cost of a routing tree is defined as the total wirelength: $cost(T) = \sum_{v \in T} |e_v|$, where e_v is the edge connecting v to its parent and $|e_v|$ is the wirelength of e_v. The algorithm consists of a bottom-up phase and a top-down phase.

In the bottom-up phase, the DME algorithm computes merging segment ms_v, representing the set of possible placement of v yielding a min-cost ZST rooted at v. The merging segment is always a *Manhattan arc*, i.e., a segment having slope $+1$ or -1. Let p be the parent of u and v. The computation of ms_p is based on ms_u and ms_v. That is, ms_p is a set of points such that the ZSTs rooted at ms_u and the ZSTs rooted at ms_v can be merged with minimum added cost (wirelength) $|e_u|+|e_v|$.

The set of points within a fixed distance of a Manhattan arc is called a *tilted rectangular region (TRR)*. The boundary of a TRR is composed of Manhattan arcs. The Manhattan arc at the center of a TRR is called its *core*. The *radius* of a TRR is the Manhattan distance between its core and its boundary. It has been shown that $ms_p = trr_u \cap trr_v$, where trr_u is the TRR with core ms_u and radius $|e_u|$ and trr_v is the TRR with core ms_v and radius $|e_v|$. The construction of a merge segment is shown in Figure 1.

Let $l(v)$ be the embedding location of v. In top-down phase, the DME algorithm embeds internal node v of G on a point in $ms_v \cap trr_p$ where trr_p is the square TRR with core $l(p)$ and radius $|e_v|$. Since both bottom-up and top-down phase run in linear time, the DME algorithm has linear-time complexity. For a linear delay model, the DME algorithm is optimal, in the sense that the total wirelength and source-to-sink pathlength are both minimum.

3 Concurrent Clock Routing and Wire Sizing

The DME algorithm does not consider the possibility of wire sizing to obtain a better zero-skew clock tree. Wire sizing are usually performed as a post-processing step.

The difficulty of keeping clock routing and wire sizing as two distinct steps while achieving a zero-skew clock tree with non-trivial wire-sizing solution is illustrated in

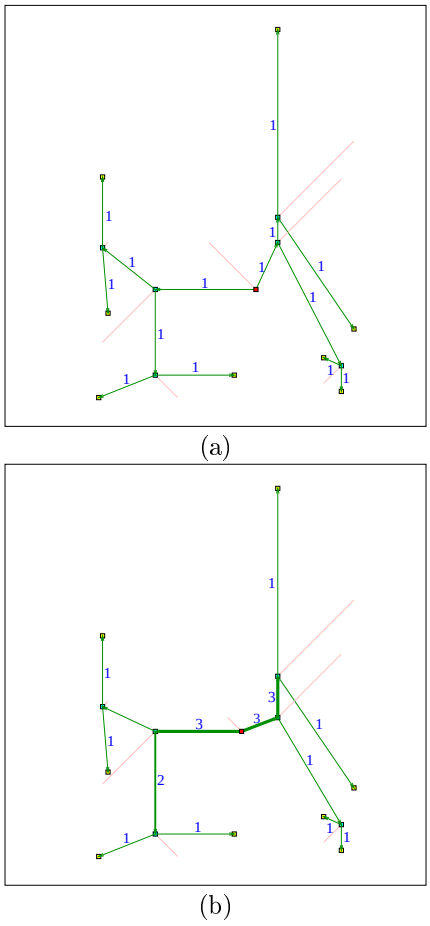

Figure 2. A clock routing problem with 8 receivers in the area of $7000 \times 7000 \mu m^2$; feasible wire widths are 1, 2 and 3× the width of a min size wire. Zero-skew clock trees generated by (a) DME + wire sizing: delay= $165ps$, wirelength = $21420\mu m$ (There is no non-trivial wire-sizing solution yielding zero-skew without changing the clock routing.); (b) IDME: delay = $119ps$, wirelength = $20869\mu m$.

Figure 2. In the clock routing problem, there are 8 receivers in the area of $7000 \times 7000 \mu m^2$. The feasible wire widths are 1, 2 and 3× the width of a min size wire. The clock tree shown in Figure 2(a) is generated by DME, following by wire sizing. Given the clock routing generated by DME, there is no non-trivial wire-sizing solution yielding zero-skew clock tree without changing the clock routing.

Figure 2(b) shows the clock tree generated by integrated clock routing and wire sizing. We made the observation that, in IDME, when wire widths are considered as design variables, merging segments can be "shifted" by varying wire widths. In other words, by considering various wire widths, we increase the number of merging points, which in turns enlarges the set of candidate embedding points. This helps shorten total wirelength. Additionally, wire sizing also reduces root-to-leaf phase delay when wires are properly tapered. In the following, we give details of the construction of a zero-skew clock routing with consideration of wire widths.

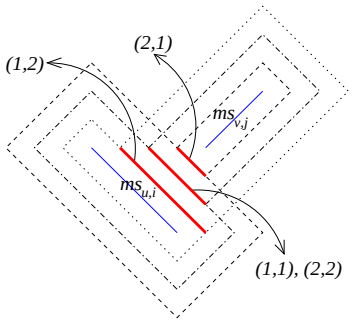

Figure 3. Formation of an MSS.

3.1 The Concept of Merging Segment Set

We extend the concept of *merging segment* to that of *merging segment set (MSS)*. An MSS is a set of merging segments, each point on which is a candidate embedding location. We show by induction that zero-skew merging points, with the consideration of various wire widths, form a set of merging segments. Assume that the merging points for u form an MSS and the merging points for v form an MSS. Let p be the parent of u and v. Given a merging segment $ms_{u,i}$ from mss_u and a merging segment $ms_{v,j}$ from mss_v, the intersection of $trr_{u,i}$ and $trr_{v,j}$ forms a TRR. This TRR degenerates to a segment when the sum of the radius of $trr_{u,i}$ and the radius of $trr_{v,j}$ is equal to the distance between $ms_{u,i}$ and $ms_{v,j}$ and $trr_{u,i}$ and $trr_{v,j}$ has the same delay. This is true for any wire width assignment. Therefore, the zero-skew merging points for p forms an MSS (see Figure 3).

3.2 Computation of MSS

Let r_e be the resistance of e, c_e be the capacitance of e, and C_e be the downstream capacitance of e. The Elmore delay from a root u to a leaf v is

$$\sum_{e \in Path(u,v)} r_e \cdot (c_e/2 + C_e)$$

Let R be the unit resistance and C be the unit capacitance. Let $d_{u,i}$ be the delay from merging segment $ms_{u,i}$ to the receivers in the subtree and $c_{u,i}$ be the downstream capacitance in the subtree seen at $ms_{u,i}$. Since we require zero-skew, the delays from the merging segment at node p to the leafs in the subtree are equal. Hence,

$$d_{u,i} + R \cdot |e_{u,i}| \cdot (C \cdot |e_{u,i}|/2 + c_{u,i}) = d_{v,j} + R \cdot |e_{v,j}| \cdot (C \cdot |e_{v,j}|/2 + c_{v,j}),$$

where $|e_{u,i}|$ and $|e_{v,j}|$ are length of $e_{u,i}$ and $e_{v,j}$, respectively.

Let k be the wirelength required for merging $ms_{u,i}$ and $ms_{v,j}$. The merging distance from $ms_{u,i}$ to the merging segment at p, using wires of sizes w_1 and w_2 for $ms_{u,i}$ and $ms_{v,j}$, respectively, is

$$|e_{u,i}| = \frac{d_{v,j} - d_{u,i} + R \cdot k \cdot (c_{v,j}/w_2 + C \cdot k/2)}{R \cdot (c_{u,i}/w_1 + c_{v,j}/w_2 + C \cdot k)}$$

and the merging distance from $ms_{v,j}$ is $k - |e_{u,i}|$.

Figure 3 shows the formation of an MSS. We enumerate all combinations of wire widths and bottom-up generate merging segments exhaustively. Therefore, the size of MSS grows exponentially with the size of subtree.

3.3 Pruning and Sampling

To keep the number of merging segments tractable, we only keep a *non-delay-inferior (NDI)* subset of merging segments. We consider the merging segment $ms_{v,i}$ to be *inferior* if there is a merging segment $ms_{v,j}$ such that

$$d_{v,i} \geq d_{v,j} \;\; and \;\; c_{v,i} > c_{v,j}$$

or

$$d_{v,i} > d_{v,j} \;\; and \;\; c_{v,i} \geq c_{v,j}$$

After pruning, an MSS contains only non-delay-inferior merging segments. However, the size of an NDI/MSS can still be large when most merging segments are not inferior. We further control the size of an NDI/MSS by *sampling* — we select some of the merging segments for representing the original NDI/MSS.

All merging segments in an MSS have the same slope; it is either $+1$ or -1. We rotate the merging segments such that all points on each of the merging segments have the same x value. If the slope of the merging segments is $+1$, we rotate the merging segments by $+\pi/4$; otherwise we rotate the merging segments by $-\pi/4$. We denote the x value of a merging segment $ms_{v,i}$ in the new coordinate system by $h_{v,i}$, called the *horizontal characteristic (hc)* of $ms_{v,i}$.

We sort all the merging segments in an NDI/MSS according to their *hc* values. Thus, the difference in *hc* of neighboring merging segments is minimum after sorting. Let $ms_{v,0}, ms_{v,1}, \cdots, ms_{v,n-1}$ be the merging segments and $h_{v,0} \leq h_{v,1} \leq \cdots \leq h_{v,n-1}$. We form a cluster for each of the merging segments.

We measure the difference between clusters by *cluster merging cost (cmc)*. We use the merging cost to decide which merging segments are to be combined. Let $ms_{v,i}$ be the merge segment in cluster a that has largest *hc* value, $ms_{v,j}$ be the merge segment in cluster b that has smallest *hc* value, and $h_{v,i} \leq h_{v,j}$. The merging cost between clusters a and b

$$cmc(a,b) = \alpha \cdot |d_{v,i} - d_{v,j}| + \beta \cdot |c_{v,i} - c_{v,j}| + \gamma \cdot |h_{v,i} - h_{v,j}|$$

where α, β and γ are the weights corresponding to each term.

We iteratively merge clusters by their *cmc* value. At each iteration, the pair of neighboring clusters having the minimum *cmc* are chosen and merged into one cluster. Each iteration will reduce the number of clusters by 1. Thus, after m iterations, there is a total of $n-m$ clusters left and each of the clusters contains one or more merging segments.

The computation of initial *cmc* values between the neighboring merging segments has complexity $O(n)$, although the sorting is of $O(n \log n)$. After each cluster merging, we only need to update the *cmc* value for the newly formed cluster. Therefore, the complexity of the iterative cluster merging is $O(n \log n)$ in the number of merging segments under sampling. After cluster merging, one merging segment in each cluster is selected to represent the merging segments in the cluster.

4 Integrated Levelized Buffer Insertion

Buffer insertion has been effective in reducing wire delay. However, in practice, buffer insertion and its location are dictated by designers. One major reason for this is that a precise characterization of buffer delay is not easy. In deep submicron design, process variation contributes significantly to device parameters, making the problem of modeling buffer delay more difficult.

In a practical solution of buffer insertion, it is usually desired that all root-to-leaf paths have the same number of buffer stages ([4]). Under this restriction, if there is a change in buffer delay due to process variation, the impact of process variation on the skew of a clock net is minimized. This is because the increment (or decrement) of phase delay is the same for all paths, although the phase delay may change.

4.1 Categorization and Promotion of Merging Segments

We refer to a buffering as *levelized* if for all paths (from a root to its leaves), the number of buffer stages is equal. For a such buffering, we can categorize merging segments according to their level. We denote the subset of merging segments of level i in the MSS at v by mss_v^i. For example, mss_v^0 consists of the merging segments on any path from which to the leafs in the subtree rooted at v there is no buffer; mss_v^3 consists of the merging segments on any path from which to the leafs in the subtree there are 3 buffers.

We say that a merging segment is *promoted* if the number of buffer stages from the merging segment to the leafs in the subtree is increased by one. Let R_b be the resistance, C_b the capacitance, and τ_b the intrinsic delay of a buffer. Let $ms_{v,j}$ be a merging segment in mss_v^i. Let $k = size(mss_v)$ be the current number of merging segments in mss_v. We define the promotion operator on $ms_{v,j}$ as performing the following: (1) create a new merging segment $ms_{v,k}$, which is a duplication of $ms_{v,j}$, followed by the updates

$$d_{v,k} \leftarrow d_{v,j} + R_b \cdot c_{v,j} + \tau_b; \quad c_{v,k} \leftarrow C_b$$

(2) add $ms_{v,k}$ to mss_v^{i+1}.

4.2 Computation of MSS under Level Restriction

Let p be the parent of u and v. We form mss_p^i from mss_u^i and mss_v^i as follows:

We compute the merging segments in mss_p^i from the merging segments in mss_u^i and the merging segments mss_v^i, following the computation described in Section 3.2. Let $max\_level(mss_p)$ be the highest buffer level in mss_p. We have the property that $max\_level(mss_p)$ equals $min\{max\_level(mss_u), max\_level(mss_v)\}$.

After the MSS at p is formed, we perform a series of promotion operations on it. That is, we update the MSS at p as follows:

1. $i \leftarrow max\_level(p)$;
2. promote $ms_{p,j}$ in mss_p^i for all j;
3. $i \leftarrow i - 1$;
4. repeat $2 - 4$ if $i \geq 0$;

After the promotion operations, we apply pruning and sampling, by the same technique described in Section 3.3, to each mss_p^i independently. At the completion of pruning and sampling, we obtain a subset of *NDI/MSS*, representing the original *NDI/MSS* under buffer level considerations

5 Experimental Results

We implemented our algorithm in C++ and experimented with it on a 64MB Celeron-366 running Linux. We ran experiments on benchmark clock problem *primary1-2* and *r1-5* [5]. The parameters of clock buffers are those used in [22], i.e., the resistance is 122Ω, the capacitance is $400fF$, and the intrinsic delay is $75ps$. The clock driver has 10Ω resistance. We generate all clock topologies by the same recursive top-down bipartition procedure in all experiments. Thus, we exclude the effects of clock topology on clock routing.

The results of clock trees generated by our implementation of DME and our algorithm (IDME) with wiresizing only are shown in Tables 1 and 2, respectively. In this experiment, we ran our algorithm in a *wirelength control* mode; that is, in our algorithm we only look for clock routings that have shorter total wirelength than that those generated by DME. The results show that the phase delay of our wiresized clock routings is uniformly much less than the phase delay generated by DME ($> 50\%$ improvement). Furthermore, we use less wirelength. (Note that no post-DME wire sizing algorithm can reduce wirelength) The increase in total wire capacitance is small, indicating that we compromise little in clock area. The CPU times required by our procedure are all small and grow roughly linearly with the problem size.

We show, in Table 3, the parameters of buffered clock routings and, in Table 4, the key parameters of buffered and wiresized clock routings, both generated by our algorithm. We set the upper limit on buffer levels to be 5 while the actual number of buffer levels and buffer locations are decided by IDME. It is seen that, with integrated buffer insertion, clock delay can be reduced by over $5\times$. In both experiments, we do not control wirelength so the delay improvements over those in Table 2

	Delay	Wirelen	Wirecap	CPU
primary1	4.00667	0.14782	3.99124	0.09
primary2	14.4379	0.40164	10.8445	0.11
r1	2.15246	1.67381	33.4762	0.09
r2	5.64462	3.46299	69.2599	0.10
r3	8.31527	4.38931	87.7862	0.16
r4	23.7507	8.80809	176.162	0.33
r5	40.5215	12.9895	259.790	0.48

Table 1. The delay (in ns), total wirelength (in $10^5\mu m$) and total wire capacitance (in pF) of zero-skew clock routings generated by the DME algorithm. The CPU times are in *seconds*.

	Delay	Wirelength	Wirecap	CPU
primary1	2.29696	0.14492	6.46314	16.14
primary2	6.84976	0.39607	13.0576	48.15
r1	1.25568	1.66560	36.1948	12.81
r2	3.01189	3.44052	74.3950	23.02
r3	3.95182	4.37204	92.5517	49.61
r4	10.6185	8.78815	184.361	129.74
r5	18.7306	12.9860	269.103	189.67

Table 2. The delay (in ns), total wirelength (in $10^5\mu m$) and total wire capacitance (in pF) of *wiresized* zero-skew clock routings generated by our algorithm (with wirelength control). The CPU times are in *seconds*.

are from performing buffer insertion (and wire sizing) simultaneously during clock routing (i.e., longer total wirelength but less delay).

The best results are obtained by simultaneously performing clock routing, wire sizing and buffer insertion. Comparing Table 4 to Table 3, we see that, with integrated wire sizing (together with buffer insertion), the clock delay is further reduced by another $10 - 15\%$, and the total wirelength is less. Due to the wire tapering, each buffer layer can drive a set of sub-trees with longer depth. Thus, the number of buffer levels is less, which in turns significantly reduces buffer usage. Again, all buffered and wiresized clock trees generated by our algorithm by construction are zero-skew (and none of known methods can guarantee so). The clock area penalty and CPU are acceptable, considering the significance of the performance gain.

6 Conclusion

We have presented an integrated deferred-merge embedding (IDME) algorithm to take into account wire sizing and buffer insertion during the bottom-up phase. This algorithm allows a trade-off between wirelength and phase delay while the impact on wire capacitance (hence power consumption) is small. Thanks to buffer insertion, phase delay can be reduced drastically with limited increase of power. We use a simple recursive top-down bipartition to generate the clock connection topology. In this way, we can clearly see that the improvement does come from the added freedom provided by wire sizing and buffer insertion. However, further wirelength improvement can be achieved if the nearest neighbor selection or clustering-based algorithm [7] is applied. Likewise, if bounded skew is considered, further improvement is expected and it is currently under our investigation.

	Delay	Wirelen	Wirecap	#l(#b)	CPU
primary1	0.92563	0.31861	8.60263	2(103)	0.95
primary2	1.17102	0.71610	19.3349	3(319)	3.43
r1	1.01314	1.93617	38.7234	3(78)	0.67
r2	1.52630	3.97432	79.4865	4(125)	1.61
r3	1.43998	4.79690	95.9380	3(123)	2.22
r4	2.00596	9.62769	192.554	4(199)	7.50
r5	2.27939	14.2709	285.419	5(390)	13.44

Table 3. The delay (in ns), total wirelength (in $10^5 \mu m$), total wire capacitance (in pF), number of buffer levels (#l) and total buffer number (#b) of *buffered* zero-skew clock routings generated by our algorithm (without wirelength control). The CPU times are in *seconds*.

	Delay	Wirelen	Wirecap	#l(#b)	CPU
primary1	0.92702	0.30534	9.43268	2(89)	62.98
primary2	1.07575	0.66343	23.0064	2(160)	222.62
r1	0.85901	1.82897	42.0810	1(31)	98.42
r2	1.28531	3.93885	82.3683	3(108)	174.95
r3	1.33653	4.78959	98.1449	3(137)	282.20
r4	1.86527	9.60415	195.610	3(183)	512.20
r5	2.11580	14.2085	288.768	4(319)	791.26

Table 4. The delay (in ns), total wirelength (in $10^5 \mu m$), total wire capacitance (in pF), number of buffer levels (#l) and total buffer number (#b) of *buffered and wiresized* zero-skew clock routings generated by our algorithm (without wirelength control). The CPU times are in *seconds*.

References

[1] N. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "Graph Algorithms for Clock Schedule Optimization," *Proc. Intl. Conf. on Computer-Aided Design*, pp. 132–136, 1992.

[2] J. L. Nerves and E. G. Friedman, "Design Methodology for Synthesizing Clock Distribution Network Exploiting Nonzero Localized Clock Skew," *IEEE Trans. on VLSI*, pp. 286–291, 1996.

[3] P. Zarkesh-Ha, T. Mule, and J. D. Meindl, "Characterization and Modeling of Clock Skew with Process Variation," *Proc. Intl. Symposium on Circuits and Systems*, pp. 441–444, 1999.

[4] J. G. Xi and W. W.-M. Dai, "Buffer Insertion and Sizing Under Process Variations for Low Power Clock Distribution," *Proc. of the Design Automation Conf.*, pp. 383–388, 1995.

[5] R. S. Tsay, "Exact Zero Skew," *Proc. Intl. Conf. on Computer-Aided Design*, pp. 336–339, 1991.

[6] T.-H. Chao, Y.-C. Hsu, J.-M. Ho, K. D. Boese, and A. B. Kahng, "Zero Skew Clock Routing with Minimum Wirelength," *IEEE Trans. Circuits Syst.-II*, pp. 799–814, 1992.

[7] M. Edahiro, "A Clustering-Based Optimization Algorithm in Zero-Skew Routing," *Proc. of the Design Automation Conf.*, 1993.

[8] A. B. Kahng and C.-W. A. Tsao, "Planar-DME: A Single-Layer Zero-Skew Clock Tree Routing," *IEEE Trans. Computer-Aided Design*, pp. 8–19, 1996.

[9] Q. Zhu and W.-M. Dai, "Planar Clock Routing for High Performance, Chip and Package Co-Design," *IEEE Trans. VLSI*, pp. 210–226, 1996.

[10] J.-H. Huang, A. B. Kahng, and C.-W. A. Tsao, "On the Bounded-Skew Routing Tree Problem," *Proc. of the Design Automation Conf.*, pp. 508–513, 1995.

[11] J. Cong, A. B. Kahng, C.-K. Koh, and C.-W. A. Tsao, "Bounded-Skew Clock and Steiner Routing Under Elmore Delay," *Proc. Intl. Conf. on Computer-Aided Design*, pp. 66–71, 1995.

[12] J.-H. Huang, A. B. Kahng, and C.-W. A. Tsao, "Bounded-Skew Clock and Steiner Routing," *ACM Transaction on Design Automation of Electronic Systems*, pp. 341–388, 1998.

[13] A. B. Kahng and C.-W. A. Tsao, "More Practical Bounded-Skew Clock Routing," *Proc. of the Design Automation Conf.*, pp. 594–599, 1997.

[14] N.-C. Chou and C.-K. Cheng, "Wire Length and Delay Minimization in General Clock Net Routing," *Proc. Intl. Conf. on Computer-Aided Design*, pp. 552–555, 1993.

[15] J. Oh, I. Pyo, and M. Pedram, "Constructing Lower and Upper Bounded Delay Routing Trees using Linear Programming," *Proc. of the Design Automation Conf.*, pp. 401–404, 1996.

[16] Q. Zhu, W. W.-M. Dai, and J. G. Xi, "Optimal Sizing of High Speed Clock Networks Based on Distributed RC and Transmission Line Models," *Proc. Intl. Conf. on Computer-Aided Design*, pp. 628–633, 1993.

[17] S. Pullela, N. Menezes, and L. T. Pillage, "Reliable Non-Zero Skew Clock Tree Using Wire Width Optimization," *Proc. of the Design Automation Conf.*, pp. 165–170, 1993.

[18] Q. Zhu, W. W.-M. Dai, and J. G. Xi, "High-Speed Clock Network Sizing Optimization Based on Distributed RC and Lossy RLC Interconnect Models," *IEEE Trans. Computer-Aided Design*, pp. 1106–1118, 1996.

[19] R. Kay and L. T. Pillage, "EWA: Efficient Wireng-Sizing Algorithm for Signal Nets and Clock Nets," *IEEE Trans. Computer-Aided Design*, pp. 40–49, 1998.

[20] S. Dhar, M. A. Franklin, and D.-F. Wong, "Reduction of Clock Delays in VLSI Structures," *Proc. Intl. Conf. on Computer Design*, pp. 778–783, 1984.

[21] B. Wu and N. A. Sherwani, "Effective Buffer Insertion of Clock Tree for High-Speed VLSI Circuits," *Microelectronics Journal*, pp. 291–300, 1992.

[22] S. Pullela, N. Menezes, J. Omar, and L. T. Pillage, "Skew and Delay Optimization for Reliable Buffered Clock Tree," *Proc. Intl. Conf. on Computer-Aided Design*, pp. 556–562, 1993.

[23] J. Chung and C.-K. Cheng, "Skew Sensitivity Minimization of Buffered Clock Tree," *Proc. of the Design Automation Conf.*, pp. 280–283, 1994.

[24] S. Pullela, N. Menezes, and L. T. Pillage, "Post-processing of clock trees via wiresizing and buffering for robust design ," *IEEE Trans. Computer-Aided Design*, pp. 691–701, 1996.

[25] A. D. Mehta, Y.-P. Chen, N. Menezes, D.-F. Wong, and L. T. Pillage, "Clustering and Load Balancing for Buffered Clock Tree Synthesis," *Proc. Intl. Conf. on Computer Design*, pp. 217–223, 1997.

[26] Y.-P. Chen and D.-F. Wong, "An Algorithm for Zero-Skew Clock Tree Routing with Buffer Insertion," *European Design and Test Conference*, pp. 230–236, 1996.

[27] G. Ellis, L. T. Pillage, and R. A. Rutenbar, "A Hierarchical Decomposition Methodology for Multistage Clock Circuits," *Proc. Intl. Conf. on Computer-Aided Design*, pp. 266–273, 1997.

[28] A. Vittal and M. Marek-Sadowska, "Power Optimal Buffered Clock Tree Design," *Proc. of the Design Automation Conf.*, pp. 230–236, 1996.

[29] G. Ellis, L. T. Pillage, and R. A. Rutenbar, "A Hierarchical Decomposition Methodology for Single-Stage Clock Circuits," *Proc. Custom Integrated Circuits Conf.*, pp. 115–118, 1997.