Clock Tree Synthesis under Aggressive Buffer Insertion

Ying-Yu Chen, Chen Dong, Deming Chen

Department of Electrical and Computer Engineering,
University of Illinois at Urbana-Champaign, Urbana, IL 61801

{chen150, cdong3, dchen}@illinois.edu

ABSTRACT

In this paper, we propose a maze-routing-based clock tree routing algorithm integrated with buffer insertion, buffer sizing and topology generation that is able to consider general buffer insertion locations in order to achieve robust slew control. Buffer insertion along routing paths had been mostly avoided previously due to the difficulty to maintain low skew under such aggressive buffer insertion. We develop accurate timing analysis engine for delay and slew estimation and a balanced routing scheme for better skew reduction during clock tree synthesis. As a result, we can perform aggressive buffer insertion with buffer sizing and maintain accurate delay information and low skew. Experiments show that our synthesis results not only honor the hard slew constraints but also maintain reasonable skew.

Categories and Subject Descriptors

10.4 [**Physical Design and Manufacturability**]: Automated synthesis of clock networks

General Terms

Algorithms, Design, Performance, Reliability

Keywords

Clock Tree, Buffer Insertion, Buffer Sizing, Maze Routing, Slew

1. INTRODUCTION

Clock distribution networks play an essential role in synchronous VLSI chips. The quality of the clock distribution network tremendously affects the performance of the chip, as the pace of almost every data transfer is determined by the clock signal. As a consequence, clock network synthesis has always been an important research topic over the years. Many works are dedicated to the problem of clock network synthesis. [1-16] are some selected works among them.

Clock tree synthesis requires accurate timing analysis in order to control clock skews among different parts of the clock tree. Also, buffer insertion is an essential part in practical clock networks, since it helps reduce delay and slew. Some clock synthesis works generate unbuffered clock networks that require a separate buffer insertion stage using conventional or specialized buffer insertion algorithms. In other works [5, 7, 9, 12-13, 15], buffer insertion and clock tree routing are integrated rather than performed separately. Therefore, the delay information can be constantly updated to guide the routing and make the clock tree more balanced.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC'10, June, 2010, Anaheim, CA, USA.

Copyright 2010 ACM 1-58113-000-0/00/0010…$10.00.

In [6, 14, 15], buffer insertion is performed in order to maintain reasonable slew in the clock network. However, potential buffer insertion locations are restricted only to the merge nodes in the clock tree topology. Given more general scenarios, such buffer arrangement may not be sufficient to meet a hard slew constraint. Figure 1 is plotted based on SPICE simulation. We can observe that the slew grows dramatically as wire length increases. Increasing driving buffer size only provides slight improvement. Therefore, buffer sizing alone cannot solve the slew control problem. For clock tree design in large chip area, buffers have to be inserted into wire segments instead of just on merge points. To the best of our knowledge, [6] is the only work that inserts buffers along the routing paths rather than solely on the merge points.

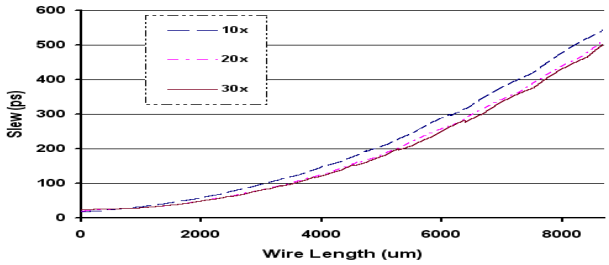

Figure 1. Buffer sizing is not sufficient to control slew.

The reason why buffer insertion along routing paths was not studied intensively is due to the difficulty of maintaining accurate delay information in the bottom-up clock tree synthesis process. For example, the slew at the input of the buffer cannot be predicted based on the downstream circuit. However, the input slew affects delay and slew of the downstream circuit severely. Buffer intrinsic delay is especially sensitive to input slew and can vary significantly when the input slew changes. Without accurate delay information, a clock tree with low skew cannot be achieved.

Our proposed algorithm is designed to balance the clock tree while allowing buffers to be inserted along routing paths. Meanwhile, buffer insertion and buffer sizing are guided by slew so that the slew in the entire clock tree is controlled under certain constraint. The contributions of our work are summarized as follows: 1) We built a library for buffers and wires with accurate delay and slew characterization, which matches the SPICE simulation results closely. 2) We guarantee bounded slew for the whole clock tree through buffer insertion and buffer sizing along the routing paths. 3) We maintain low skew through accurate timing analysis and balanced routing.

The rest of the paper is organized as follow: In section 2, we review the clock tree synthesis problem and some selected previous approaches. In section 3, we present the delay model we use. In section 4, we explain and analyze our algorithm in detail. Experimental results are presented in section 5, and section 6 concludes the paper.

2. BACKGROUND

Given a set of clock sinks, the general clock tree synthesis problem is to construct a clock distribution network in a binary tree topology, in which the root node represents the clock source; the leaf nodes represent the clock sinks, and the internal nodes represent the merge nodes. Clock skew is defined as the maximum difference among the delays from the clock source to all clock sinks. Latency is defined as the maximum of the delays from the clock source to all sinks. Some clock tree synthesis algorithms target to minimize the clock skew [1-2], while others bound the clock skew within a given hard constraint [8].

The Deferred-Merge Embedding (DME) algorithm [1] is the fundamental of many later clock tree works. It consists of a bottom-up stage and a top-down stage. A tree of merge segments, which represent the possible locations of merge nodes, is constructed in the bottom-up stage. Once all the merge segments are constructed, the exact locations of merge nodes are determined in the top-down stage. Merge segments are calculated to balance the delays of the two sub-trees.

Some later clock tree synthesis algorithms modify the merge segment calculation in order to capture more complicated delay models [16]. Others extend the concept of merge segments to merge regions in order to establish some flexibility during the synthesis [7-9]. However, they are only valid under the assumption that no buffers are inserted and no detours are taken in the routing paths. In practical circuits, buffer insertion along the routing path can be helpful in terms of slew control.

3. DELAY MODELING

Our goal is to match our delay and slew estimation as close to simulation results as possible while maintaining reasonable complexity. We performed a series of experiments on different delay models and on SPICE to find the characteristics of delay and slew, and based on the characteristics, we developed a delay/slew analysis scheme that is suitable for our buffered clock tree synthesis algorithm.

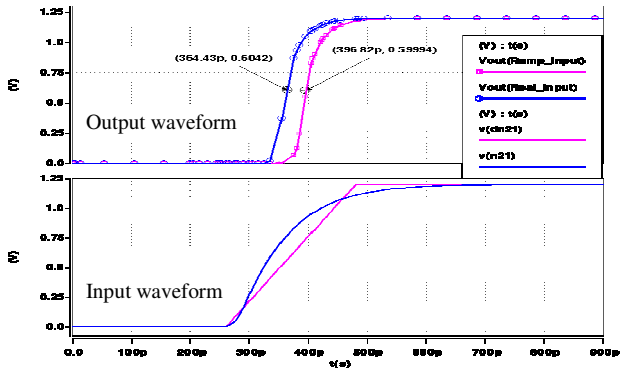

Figure 2. Error of using ramp input.

3.1 Insufficiency of existing delay models

Elmore delay is widely used in clock tree design to estimate interconnect delay due to its simple and closed-form expression. It has been widely known that Elmore delay matches the first moment of impulse response and can be highly inaccurate by ignoring resistive shielding. More importantly, Elmore delay cannot compute slew within an RC netlist. Existing works [18, 19] developed closed-form delay and slew expression of ramp input by matching higher order moments of the impulse response. However, by approximating real curve into ramp signal can introduce large amount of error. As shown on the bottom of Figure 2, the buffer input signal coming from the previous buffer is approximated into a ramp input. Although both of them have a 150ps 10%-90% slew, the buffer output signal can shift by 32ps (the top figure). This study indicates that for accurate delay estimation the buffer input waveform needs to be considered during delay propagation. Because there is no accurate model to capture slew propagation, our approach is to carry out SPICE simulation considering input slew of the driving buffer, driving buffer type, and load wire length to determine the output waveform. The output waveform can drive the next stage buffer for accurate delay and slew estimation. Because SPICE simulation is time consuming, our approach is to pre-characterize the delay and slew using the aforementioned parameters and build multi-dimensional functions. The next section explains how we carry out this characterization in details.

3.2 Delay/slew library-based implementation

We partition our clock tree into smaller components with cuts on buffered nodes. The components act as units on which we perform delay/slew estimations. The circuitry structures of the components can be categorized in two types: single wire and branch.

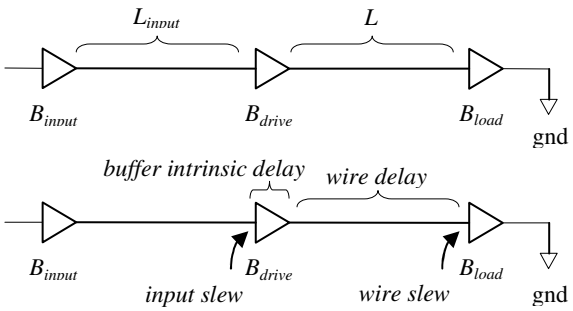

Figure 3. Circuit structure used for simulation.

For single-wire-typed components, we build small circuits as depicted in Figure 3. B_{drive} and B_{load} are the driving and load buffers, which can be of different types in the buffer library. L is the wire length. We add an extra input buffer B_{input} and a wire segment of length L_{input} in order to create the special buffer output waveform as the actual input to the buffers and wires we want to measure. The input waveform is an ideal ramp, and B_{input} transforms it into the special buffer output waveform. L_{input} can be adjusted to generate a range of input slew.

For each combination of driving buffer type and load buffer type, we sweep a range of different lengths for L_{input} and L and measure the input slew, buffer intrinsic delay, wire delay, and wire slew. Figure 3(b) shows the data we need to measure. We then use MATLAB to perform surface-fitting on the collected data, that is, buffer intrinsic delay, wire delay, and wire slew, with respect to input slew and length, and obtain a set of delay estimation functions. These functions are 3^{rd}- or 4^{th}-order polynomials in terms of input slew and length, which are sufficiently accurate for the range of input slew and lengths we need.

For branch-typed components, we consider only cases with 2 branches because the tree is binary. The data of wire delays and wire slews of both branches are collected. We perform hyperplane-fitting instead of surface-fitting on simulation data due to the increase of dimensions. The resulting polynomial functions are used similarly as in the case of single-wire-typed components.

4. BUFFERED CLOCK TREE SYNTHESIS

The problem we are to solve by our proposed algorithm can be formulated as follows: Given a set of clock sinks, different types of buffers, and a single type of wire, we want to synthesize a buffered clock tree such that the slew in any part of the clock tree is bounded, and the clock skew is minimized.

4.1 Top-level Algorithm

Our clock tree synthesis framework is a *levelized* adaption from [2]. Also, the merging stage is replaced by our proposed merge-routing algorithm. Building a levelized clock tree helps reduce the skew because the tree itself is balanced. Figure 4 illustrates our top-level algorithm.

```
Given a set of clock sinks S
BufferedClockTreeSynthesis(S)
{
   V ← S;
   G ← InitializeNearestNeighborGraph(V);
   while (|V| > 1) {
      E ← FindMatching(G);
       for each e ∈ E {
         (v1, v2) ← e.endpoints;
         M ← MergeRouting(v1, v2);
         G ← UpdateNearestNeighbors(v1, v2, M);
      }
   }
   return V;  // contains only the root node of the clock tree
}
```

Figure 4. Top-level algorithm.

Initially, a nearest-neighbor graph is constructed based on a cost function that is able to capture the feasibility of pairing the nodes and merging them into a sub-tree. The edge cost is a function of distance and delay difference between each pair of nodes. Because smaller distance and delay difference are more desirable, we have the edge cost of an edge e connecting nodes v_1 and v_2 as follows:

$$cost(e)=a \cdot distance(v_1,v_2)+\beta \cdot \left|delay(v_1)-delay(v_2)\right|$$

After the nearest-neighbor graph is constructed, we find a matching in the graph, which represents the pairs of sub-trees that are most feasible for merging in the current level. We process (*merge*) each of the edges in the matching and obtain merge nodes as new candidates for pairing. The processed edges are removed from the nearest-neighbor graph, and the new merge nodes are added into the graph. When all edges in the matching are processed, we complete building a level. Then, we find another matching in the updated nearest-neighbor graph and continue to build the next level. The algorithm terminates when only one node, which is the root of the clock tree, is left in the nearest-neighbor graph, indicating that all sinks are embedding into a single tree.

The algorithm we use to find a matching is a greedy heuristic. First, we compute the centroid of all sink coordinates. Then, we repeatedly choose the node which is farthest from the centroid and its nearest neighbor to form a pair. In the case of having odd numbers of nodes, a seed node is chosen not to be matched with other nodes and is directly transferred to the next level. The seed node is chosen as the node with maximum latency due to the fact that the nodes in the next level have larger delays. In this way, delays can be more balanced when the seed node and another node are merged in the next level.

4.2 Merge-routing Algorithm

The proposed merge-routing algorithm consists of three stages: balance, route, and binary search. The balance stage, which is a preprocessing stage, reduces the difference of delays before two sub-trees are routed. The routing stage finds two buffered routing paths that have minimum skew. The binary search stage adjusts the location of the merge node so that the skew is reduced to the minimum.

4.2.1 Balance Stage

When the top-level algorithm selects a pair of sub-trees to merge, there is limited amount of delay that merge-routing can balance without taking detours. The amount can be roughly estimated based on the distance of the roots of the sub-trees. If this amount is not sufficient to balance the delay difference, wire-snaking needs to be performed.

In order to honor the slew constraint during wire-snaking, we propose a progressive approach that inserts wires and buffers alternatively until the target delay is achieved. First, a segment of wire starting with a driving buffer is inserted. The wire length is gradually increased, until either the slew at the end of the wire reaches the slew limit or the target delay is achieved. The process repeats until the target delay is finally achieved.

4.2.2 Routing Stage

The routing algorithm adopts the concept of global routing in standard cell design. Routing area has been defined and partitioned into routing grids. For clock tree routing, all the nets are two pin nets and the length of nets varies in a wide range.

Two innovations have been made in our clock tree routing framework. First, to balance accuracy and run time of widely distributed net lengths, we dynamically adjust the routing grid size R based on distance of each pair of nodes to be routed.

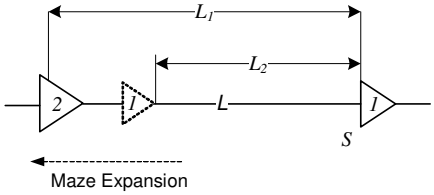

Figure 5. Maze expansion and intelligent buffer sizing.

The second innovation of this work is bi-directional maze routing. Instead of routing from one sink to another, routing starts from two sinks simultaneously. Each routing grid contains delay information from both routing. The grid with minimum delay difference is chosen as a tentative merger location. Moreover, buffers are inserted along wires to keep slew in control. A pre-characterized delay library (introduced in section 3) is used for delay/slew look-up. The procedure of maze expansion and delay library lookup is demonstrated by the example in Figure 5:

First, find the load capacitance of wire segment L. In this example, load capacitance is the gate capacitance of type 1 buffer. During routing, the length of wire segment L increases gradually. For each L, find segment propagation delay from delay library by assuming driving buffer input slew to be the slew limit. The slew at the other end of wire (S) is monitored as well. If at certain point L_2, slew S reaches the limit, one buffer needs to be inserted to restore the slew of segment L. The size of the buffer is chosen so as to make slew S closest to the slew limit.

4.2.3 *Binary Search Stage*

After the routing stage is completed, we perform binary search to find the optimum merge node that minimizes the delay difference between the two routing paths. The merge node is moved within the area between the final routing bins according to top-down timing analysis results. Because our SPICE-based timing analysis is more accurate than Elmore delay model and is able to handle buffer delays, this method outperforms the merge node calculation method in [3] in terms of accuracy.

Table 1. Experimental results of GSRC benchmarks.

		Our Results			Comparison		
	# of Sinks	Worst Slew (ps)	Skew (ps)	Max Latency (ns)	Skew [12] (ps)	Skew [5] (ps)	Skew [15] (ps)
r1	267	89.5	69.7	1.30	100	57.0	37.0
r2	598	89.3	59.9	1.69	96	87.4	59.5
r3	862	89.7	64.2	1.95	101	59.6	49.5
r4	1903	100.0	107.1	2.75	176	98.6	59.8
r5	3101	98.3	89.4	3.00	110	86.9	50.6

Table 2. Experimental results of ISPD benchmarks.

	# of Sinks	Worst Slew (ps)	Skew (ps)	Max Latency (ns)
f11	121	99.2	45.2	2.26
f12	117	83.6	45.8	1.92
f21	117	99.2	51.1	2.16
f22	91	100.0	42.4	1.62
f31	273	98.1	65.1	4.22
f32	190	85.2	52.3	3.38
fnb1	330	80.0	68.6	4.67

5. EXPERIMENTAL RESULTS

We implemented our framework in C++. We used 45nm PTM model as technology parameters. We performed experiments on benchmarks from the GSRC Bookshelf [17], which are the benchmarks used in most of the clock network synthesis works [1-5, 7-10, 12-16]. We used a library of 3 buffers, which are defined in transistor level using SPICE. Unit capacitance and unit resistance are 0.03 Ω/μm and 0.2fF/μm, which are 10X bigger than what are specified in the benchmarks of [17]. This mimics bigger chips that incur stringent slew constraints. In this way, delay and especially slew grow much faster with the increase of wire length, and the need to insert buffers along routing paths is emphasized. Therefore, it is more suitable to test our algorithm in this setting, yet it is still reasonable to compare the quality of our work with that of exiting works.

The experimental results are shown in Table 1. The slew limit is set as 100ps, and we set it to 80ps during synthesis in order to leave a margin. The run time for every benchmark is within several minutes. The worst slew, the skew, and the maximum latency are obtained from SPICE simulation of the clock tree netlist. The worst slew is the maximum slew among all nodes in the clock tree reported by SPICE. For every benchmark, it does not exceed the slew limit of 100ps. It is the best to compare our results with [6], since it also inserts buffers in the clock tree routing paths. However, the benchmarks used in [6] are not available to us. Some other works that integrates buffer insertion at the merge points in clock tree synthesis are [5, 12, 15], which are also suitable for comparison. Despite the fact that we have a much more difficult task of skew reduction and slew control by using 10X unit resistance and capacitance, our skew results are comparable to those in [5, 12, 15]. The skews of [5, 12, 15] are extracted from what are reported in [15].

We also performed experiments on the benchmarks from the ISPD2009 Contest. These benchmarks have large areas (~1cm) with respect to unit RC (~10^{-4} ohm/nm, fF/nm), similar to our 10X RC setting for the GSRC benchmarks, and it requires buffer insertion along wires to control slew. Originally, these benchmarks contain obstacles, but we took them out since it was not the focus of our work. Table 2 shows the results. The run time is also within minutes. It can be noted that all skews are within 3% of maximum latency. Other works have not handled these benchmarks so we do not have comparison results.

6. CONCLUSION

We have proposed a buffered clock tree synthesis algorithm. The potential buffer locations are not limited to merge nodes, and thus we are able to have robust slew control. Furthermore, we have overcome the difficulty and inaccuracy of delay and slew in a bottom-up buffer insertion scheme. The skew of our clock tree remains reasonable under such aggressive buffer insertion.

7. ACKNOWLEDGMENTS

This work is partially supported by the NSF grant CCF 07-02501, SRC grant 2007-HJ-1592, and a grant from Sun Microsystems, Inc.

8. REFERENCES

[1] T.-H. Chao, Y.-C. Hsu, J.-M. Ho and A. B. Kahng, "Zero skew clock routing with minimum wirelength," *IEEE Trans. Circuits Syst. II, Analog Digit. Signal Process,* vol. 39, pp. 799-814, Nov. 1992.

[2] M. Edahiro, "A Clustering-Based Optimization Algorithm in Zero-Skew Routings," *DAC 1993,* pp. 612-616, 1993..

[3] R. -S. Tsay, "An exact zero-skew clock routing algorithm," *IEEE Transactions on CAD,* vol. 12, pp. 242-249, Feb. 1993.

[4] J. Chung and C.-K. Cheng, "Optimal buffered clock tree synthesis," *Intl. ASIC Conference and Exhibit 1994 ,* pp. 130-133, 1994.

[5] Y. P. Chen and D. F. Wong, "An algorithm for zero-skew clock tree routing with buffer insertion," *ED&TC 1996,* pp. 230-236, 1996.

[6] G. E. Tellez and M. Sarrafzadeh, "Minimal buffer insertion in clock trees with skew and slew rate constraints," *IEEE Transactions on CAD,* vol. 16, pp. 333-342, 1997.

[7] A. Vittal and M. Marek-Sadowska, "Low-power buffered clock tree design," *IEEE Transactions on CAD,* vol. 16, pp. 965-975, 1997.

[8] J. Cong, et al., "Bounded-skew clock and Steiner routing," *ACM Trans.Des.Autom.Electron.Syst.,* vol. 3, pp. 341-388, 1998.

[9] I. Liu, T. Chou, A. Aziz and D. F. Wong, "Zero-skew clock tree construction by simultaneous routing, wire sizing and buffer insertion," *ISPD 2000,* pp. 33-38.

[10] H. Su and S. S. Sapatnekar, "Hybrid structured clock network construction," *ICCAD 2001,* pp. 333-336, 2001.

[11] P. J. Restle, et al., "A clock distribution network for microprocessors," *IEEE J. Solid-State Circuits,* vol. 36, pp. 792-799.

[12] R. Chaturvedi and J. Hu, "Buffered clock tree for high quality IC design," *ISQED 2004.* pp. 381-386, 2004.

[13] J.-L. Tsai, T.-H. Chen and C. C. -P. Chen, "Zero skew clock-tree optimization with buffer insertion/sizing and wire sizing," *IEEE Transactions on CAD,* vol. 23, pp. 565-572, 2004.

[14] G. Venkataraman, et al., "Practical techniques to reduce skew and its variations in buffered clock networks," *ICCAD 2005.* pp. 592-596.

[15] A. Rajaram and D.Z. Pan, "Variation tolerant buffered clock network synthesis with cross links," *ISPD 2006,* pp. 157-164.

[16] U. Padmanabhan, J. M. Wang and J. Hu, "Robust Clock Tree Routing in the Presence of Process Variations," *IEEE Transactions on CAD,* vol. 27, pp. 1385-1397, 2008.

[17] A. B. Kahng and C.-W. A. Tsao, VLSI CAD Software Bookshelf: Bounded-Skew Clock Tree Routing, May 2000. [Accessed: Nov 18 2009] http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/BST/

[18] C. J. Alpert, et al., "Closed-Form Delay and Slew Metrics Made Easy", *IEEE Transactions on CAD*, Vol. 23, No.12, Dec. 2004.

[19] C. V. Kashyap, et al., "PERI: A Technique for Extending Delay and Slew Metrics to Ramp Inputs", *ACM symp. Physical Design*, 2003