CLOCK TREE SYNTHESIS FOR PRESCRIBED SKEW SPECIFICATIONS

A Thesis

by

RISHI CHATURVEDI

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

May 2004

Major Subject: Computer Engineering

CLOCK TREE SYNTHESIS FOR PRESCRIBED SKEW SPECIFICATIONS

A Thesis

by

RISHI CHATURVEDI

Submitted to Texas A&M University
in partial fulfillment of the requirements
for the degree of

MASTER OF SCIENCE

Approved as to style and content by:

J. Hu (Chair of Committee)	
W. Shi (Member)	U. Cilingiroglu (Member)
D. M. H. Walker (Member)	C. Singh (Head of Department)

May 2004

Major Subject: Computer Engineering

ABSTRACT

Clock Tree Synthesis for Prescribed Skew Specifications. (May 2004)
Rishi Chaturvedi, B. Tech., Indian Institute of Technology, Kanpur
Chair of Advisory Committee: Dr. Jiang Hu

In ultra-deep submicron VLSI designs, clock network layout plays an increasingly important role in determining circuit performance including timing, power consumption, cost, power supply noise and tolerance to process variations. It is required that a clock layout algorithm can achieve any prescribed skews with the minimum wire length and acceptable slew rate. Traditional zero-skew clock routing methods are not adequate to address this demand, since they tend to yield excessive wire length for prescribed skew targets. The interactions among skew targets, sink location proximities and capacitive load balance are analyzed. Based on this analysis, a maximum delay-target ordering merging scheme is suggested to minimize wire and buffer area, which results in lesser cost, power consumption and vulnerability to process variations. During the clock routing, buffers are inserted simultaneously to facilitate a proper slew rate level and reduce wire snaking. The proposed algorithm is simple and fast for practical applications. Experimental results on benchmark circuits show that the algorithm can reduce the total wire and buffer capacitance by 60% over an extension of the existing zero-skew routing method.

To my parents

ACKNOWLEDGMENTS

I am greatly indebted to my advisor, Dr. Jiang Hu, for giving me an opportunity to work under him. I would like to thank him for all the invaluable guidance he provided me throughout the course of this research. I really appreciate the freedom he gave me while working on my research.

I am thankful to Dr.Weiping Shi for valuable discussions and to Dr. G. Ellis for providing a test case with prescribed skew specifications. I would also like to express my sincere appreciation to all my committee members for their interest in my research. Last, but not the least I would like to thank my parents for their absolute confidence in me. They have always been a constant source of inspiration to me.

TABLE OF CONTENTS

CHAPTER Page

I INTRODUCTION 1

II PRELIMINARY 9

A. Models Used 10
B. Delay Balancing 11

III ALGORITHM 14

A. Review of the Deferred-Merge Embedding (DME) Algorithm 14
1. Bottom-Up Phase: Construction of the Tree of Merging Segments 14
2. Top-Down Phase: Embedding of Nodes 19
B. The Merging Scheme 22
C. Buffer Insertion 25
1. Load Constraint 25
2. Delay Balancing 26
D. Complexity 28

IV EXPERIMENTS AND RESULTS 29

A. Experimental Setup 29
B. Unbuffered Prescribed-skew Clock-tree 29
C. Buffered Prescribed-skew Clock-tree 36
D. Zero-skew Clock-tree 40

V CONCLUSION 41

REFERENCES 42

APPENDIX A 46

VITA 49

LIST OF TABLES

TABLE Page

I Comparisons of clock routing with different merging schemes under non-zero skew targets. The wirelength reduction is with respect to the wirelength from NS of [13]. 31

II Comparison of our buffered clock tree routing and an extension to the NS algorithm [13]. 37

III Wirelength from our MAT algorithm and CL algorithm in [13]. The number in each parentheses is the ratio with respect to wirelength from CL+I6 algorithm in [13]. 40

LIST OF FIGURES

FIGURE Page

1 With delay targets for four sinks $t_4 > t_3 >> t_2 > t_1$, a traditional merging scheme may result in an abstract tree in (a) and embedding in (b) with wire snakings. A different abstract tree in (c) and its layout embedding in (d) may yield less wirelength. 3

2 An example of level by level buffer insertion (a) abstract tree; (b) physical tree. 6

3 Clock tree for H-tree based internal nodes placement results in higher wire delay due to long wires connecting to the top level buffer. 7

4 An example of balanced buffer insertion (a) abstract tree; (b) physical tree. 8

5 Interconnect model. 10

6 Buffer model. 10

7 Equivalent RC circuit using the interconnect model and the buffer model. 11

8 Examples of merging subtrees without wire snaking in (a) and with wire snaking when delay-target t_j at v_j is significantly greater than delay-target t_k at v_k in (b). 12

9 Construction of merging segment with: (a) no wiresnaking (b) wiresnaking. 16

10 Construction of the tree of segments. 17

11 An example of the bottom-up construction of the merging segment tree. 18

12 Fixing the exact location of internal nodes in the zero skew clock tree. 19

13 Fix the location of v given the placement of the parent p. 20

FIGURE Page

14 An example of the top-down embedding of internal nodes. 21

15 Algorithm of the merging selection scheme. 25

16 Algorithm for applying the load constraint. 26

17 Buffer insertion to reduce wire snaking. Delay target $t_j > t_k$. 27

18 Clock tree obtained by NS. 32

19 Clock tree obtained by minimum merging cost based algorithm - MIC. 33

20 Clock tree obtained by maximum delay target based ordering algorithm - MAT. 34

21 Clock tree obtained by maximum delay target and minimum merging cost based ordering algorithm - MAT-MIC. 35

22 Buffered clock tree obtained by extended zero skew tree along with load constraint. 38

23 Buffered clock tree obtained by maximum delay target and minimum merging cost based ordering along with load constraint. 39

CHAPTER I

INTRODUCTION

The quality of a synchronous digital integrated circuit heavily depends on clock network design, especially under current ultra-deep submicron technology. First, the clock signal determines the pace of data transfer and operation frequency[1]. Second, the clock network is one of the largest nets and one of the most frequently switching nets at the same time, thus it has a paramount influence on power efficiency of the circuit. Third, due to its large size, the switching of clock signal may draw huge current from power/ground network and incur power supply noise. Last but not least, clock signal is vulnerable to process variations[2, 3] and the induced clock signal variation may in term affect circuit design and timing. Therefore, it is vitally important to have a clock layout algorithm addressing these concerns for a high quality integrated circuit design.

A clock network design usually starts with specifying delay-targets from the source to each sink which is either a flip-flop or a latch. Since clock skew, which is the difference of delay between clock sinks, is more important than the delay itself, this specifying process is often called skew scheduling. The basic objective of skew scheduling is to minimize the clock period subject to setup-time and hold-time constraints [1, 4]. After skew scheduling, the delay-targets or skew specifications are achieved through clock network routing.

It was observed long time ago that certain non-zero skew could be utilized to improve clock frequency [1, 5]. Such skew is called prescribed skew to be distinguished from any unwanted non-zero skew. In this scenario, a skew refers to the delay dif-

---

The journal model is *IEEE Transactions on Automatic Control*.

ference between a certain sink pair. In addition to timing improvement, prescribed non-zero skews also help to reduce simultaneous signal switching and power supply noise[6]. Moreover, tolerance to process variations can be improved by setting each skew value close to the center of its permissible range[7]. Therefore, prescribed non-zero skew is a very promising approach to improve circuit timing, power supply noise and reliability. Then, is there any clock network layout algorithm that can accomplish the prescribed skews with minimum size and desired slew rate? In general, a small clock network size implies less cost, less power consumption and less vulnerability to process variations.

A common structure for clock network is a routing tree where the clock source drives the root node and the clock sinks are the leaf nodes. Without loss of generality, we can conceive the clock tree routing as a process that recursively merges a set of subtrees in a bottom-up fashion. Initially, each clock sink is a subtree and then the subtrees are merged in pairs. A pair of subtrees is merged to form a new subtree whose root is the merging node. This procedure proceeds till there is only one subtree left and this single subtree is connected to the source directly. There are two major decision-makings in this clock tree routing process: (i) *merging scheme* that tells which subtrees should be merged together; (ii) *layout embedding* that decides locations for the merging nodes. The merging scheme can be extracted out and performed in advance to construct an abstract tree. The internal nodes in the abstract tree correspond to the merging nodes without specifying locations. Abstract tree construction and layout embedding can be performed either separately or in an integrated manner. Examples of abstract tree and embedding are shown in Figure 1. The figure 1(a) and figure 1 (c) are the two possible abstract trees, and figure 1 (b) and figure 1 (d) are their physical trees respectively.

Most of previous works on clock network design attempt to minimize clock skew

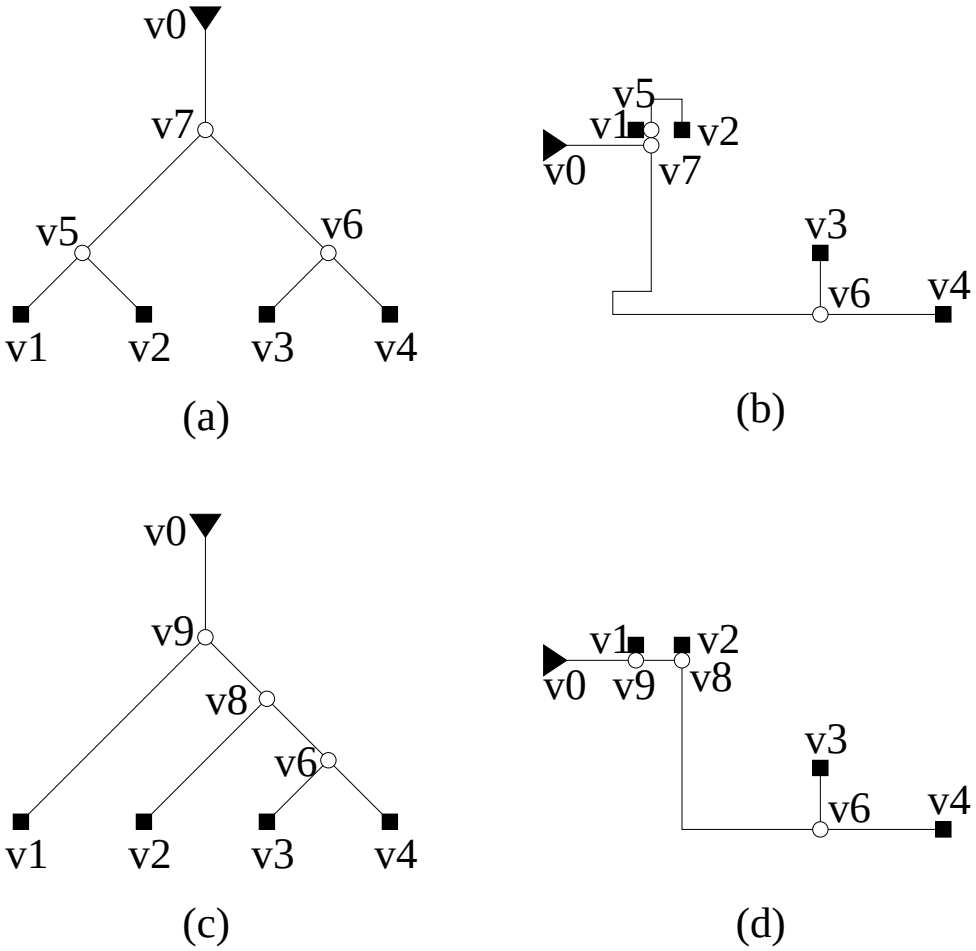


Fig. 1. With delay targets for four sinks $t_4 > t_3 >> t_2 > t_1$, a traditional merging scheme may result in an abstract tree in (a) and embedding in (b) with wire snakings. A different abstract tree in (c) and its layout embedding in (d) may yield less wirelength.

or obtain zero skew, because the skew is a lower bound for clock period time [5]. In this scenario, a more precise definition of skew is the *maximum* delay difference among all clock sinks. Early influential zero skew routing works include H-tree [5], top-down recursive partitioning [8] and bottom-up recursive matching method [9]. However, these methods emphasize on load balancing without evaluating actual delay. In [10], Tsay introduced an Elmore delay based layout embedding technique that can achieve exact zero skew for any given abstract tree. In order to further reduce wirelength,

the DME (Deferred Merge Embedding) algorithm was developed in [11] according to the observation that there are multiple locations for a merging node to satisfy skew specifications. Instead of committing a merging node to particular location immediately, DME identifies and maintains *merging segment* for each merging node in a bottom-up tree traversal. After merging segments for all merging nodes are found, a top-down tree traversal is conducted to choose one location on each merging segment such that the total wirelength is minimized. Both Tsay's embedding and DME embedding technique can be applied to achieve any non-zero skew as well. For any given abstract tree and Elmore delay model, DME is a very mature layout embedding technique to obtain any skew specifications with minimal wirelength and becomes a basis for many subsequent clock routing works [12, 13, 14, 15].

For merging schemes, a widely accepted conclusion is that a subtree should be merged with its nearest neighboring subtree to save wirelength. For early VLSI technologies, interconnect delay is dominated by capacitive load, thus many previous merging schemes [8, 9, 11] sought for a balanced abstract tree to facilitate zero skew. However, Edahiro noted in [13] that sometimes an unbalanced abstract tree might yield less wirelength even for zero skew clock routing. This is due to the fact that distributed wire RC delay started to dominate and merely balancing capacitive load is not adequate. In [13], the merging selection is integrated with DME embedding. At each step, Edahiro chose a subset (generally less than a half) of subtrees to be merged in pairs in contrast to choosing all subtrees in other works. The work of [13] reported so far the best wirelength for zero skew routing.

In contrast to numerous works on zero skew clock routing, there are very few works reported on prescribed non-zero skew routing despite its great importance. Perhaps this is due to the misconception that existing zero skew routing techniques can be applied to non-zero skew directly. Indeed, the layout embedding techniques

originally designed for zero skew [10, 11] can be adopted directly to achieve non-zero skews. However, zero skew driven *merging schemes* do not necessarily work well for non-zero skew clock routing. In fact, we discover that huge wirelength is generated through traditional merging scheme in which only subtree spatial proximity is considered while delay-target differences are ignored. This is especially true when the differences among delay-targets are large so that a lot of wire snakings [10] are incurred. The example in Figure 1 illustrates that different merging schemes (abstract trees) may provide different wirelength for non-zero skew clock routing.

A few works [14, 15] integrate skew scheduling with clock routing to exploit the useful skews. Starting with a zero skew routing tree, the work of [15] performs merging segment perturbation and gate sizing to minimize power consumption subject to setup-time and hold-time constraints for a fixed clock period time.

In [14], an incremental scheduling algorithm is proposed and combined with the DME embedding for a given abstract tree. However, skew scheduling is often carried out individually ahead of clock routing in practical design flows.

Since a clock network is normally very large, buffers are often employed to ensure an acceptable slew rate. Many previous works[16, 17, 18, 19, 20, 21, 22] place buffers of the same size at nodes of the same level in the clock tree as shown in Figure 2. This is done for two reasons: (1) zero skew routing generally results in a balanced tree; (2) this level by level buffering scheme can reduce the effect of inter-die process variations. A conventional H-tree based buffered clock-tree helps in skew minimization via its symmetry as shown in Figure 3 but the wires connecting to the top-level buffer are much more longer than the wires connecting to the flip-flops. This leads to higher wire delays according to the Elmore delay model. Figure 2 and Figure 4 which show balanced buffer insertion technique and level by level buffer insertion technique may not be applicable to non-zero skew routing as it may generate unbalanced trees.

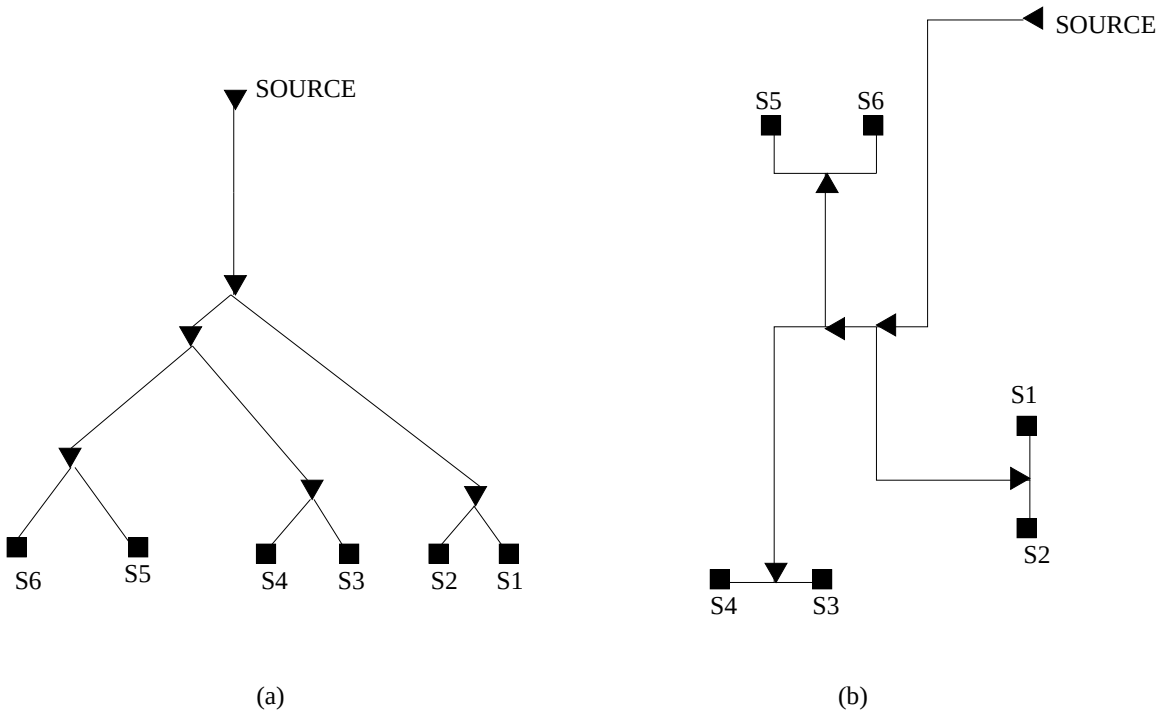


Fig. 2. An example of level by level buffer insertion (a) abstract tree; (b) physical tree.

Further, the increasingly significant intra-die process variations[23] request for a more general variation tolerant technique such as non-zero skew scheduling[7]. A buffered clock tree algorithm for prescribed skews is proposed in [24]. However, this method restricts that the prescribed skew can take only a few discrete values.

The goal of this work is to develop a clock routing algorithm that facilitates a high performance, low power, low noise and variation tolerant clock network. In this work, we analyze the interactions among skew targets, sink location proximities and capacitive load balance in clock routing. According to this analysis, a maximum delay-target based merging scheme is proposed. This merging scheme is integrated with buffer insertion and DME embedding to achieve any continuous prescribed skews. The total capacitance of wire and buffers is minimized to restrict cost, power consumption and vulnerability to process variations. Buffer insertion plays two roles here: (1)

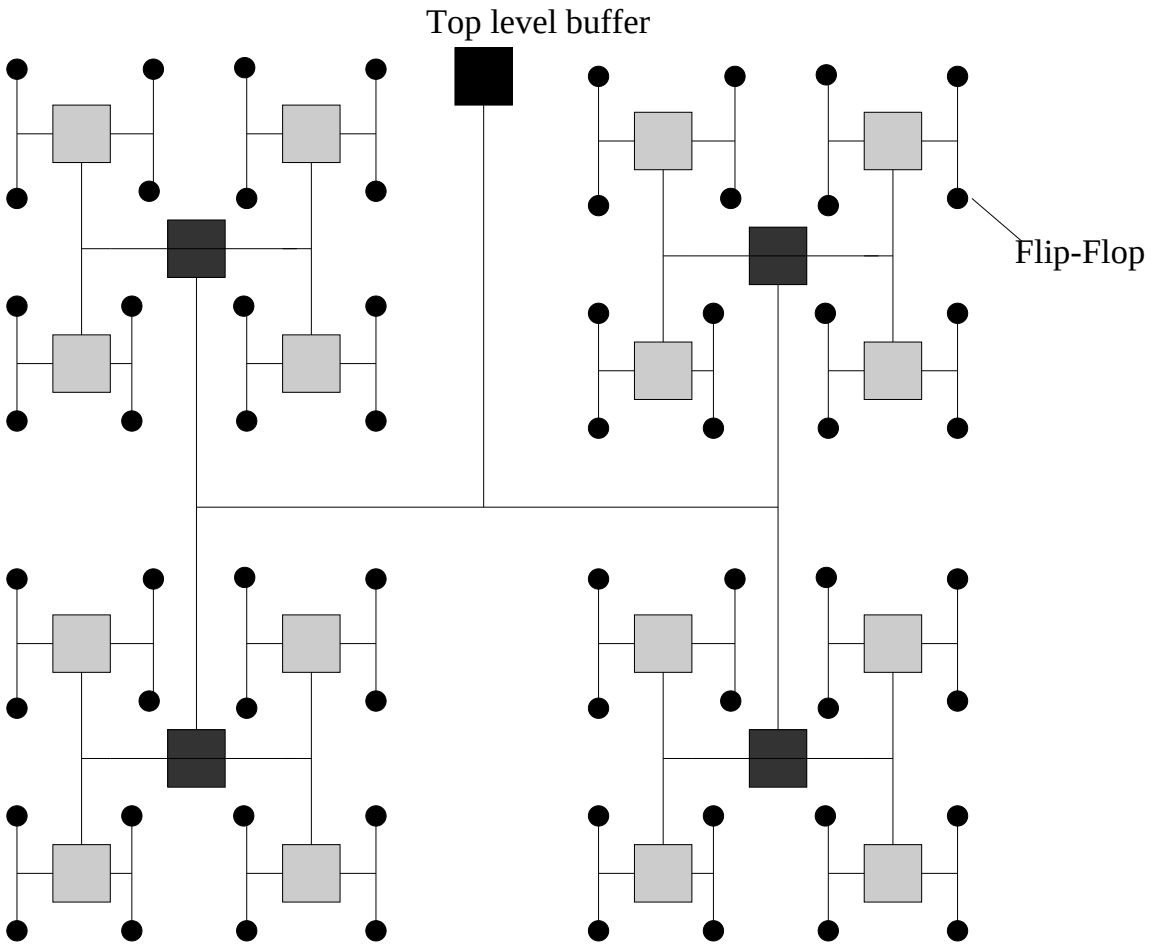


Fig. 3. Clock tree for H-tree based internal nodes placement results in higher wire delay due to long wires connecting to the top level buffer.

enforce a maximum load constraint to ensure signal slew rate; (2) reduce wire snaking by balancing delay targets.

The proposed buffered clock routing method is simple and fast for practical applications. We compared our routing method with extension to traditional zero skew clock routing method [13] on benchmark circuits. The experimental results show that our method can meet non-zero skew specifications and load capacitance constraint with 60% less wire and buffer capacitance.

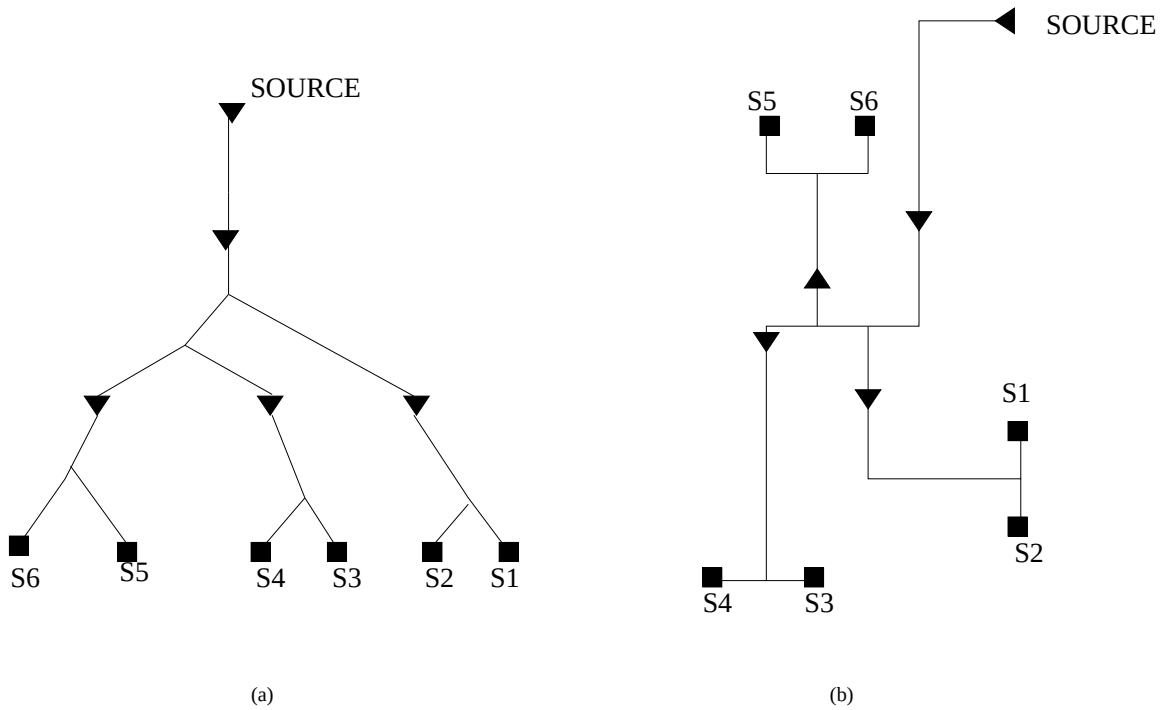


Fig. 4. An example of balanced buffer insertion (a) abstract tree; (b) physical tree.

CHAPTER II

PRELIMINARY

Same as other clock routing works, we adopt the Elmore delay model for delay computation [25]. The wire cost and buffer cost are expressed through their capacitance. The total wire and buffer capacitance is also an indication of the dynamic power consumption. The problem we will solve is formally stated as follows.

Prescribed Skew Buffered Clock Routing Problem: *Given a set of clock sinks* $V = \{v_1, v_2, ...v_n\}$*, load capacitance* C_i *for each sink* $v_i \in V$*, skew specifications* $q_{i,j}$ *for every pair of sinks* $v_i, v_j \in V$*, a buffer type b, find a buffered Steiner tree with clock sinks as leaf nodes such that the total buffer and wire capacitance is minimized, the skew specification* $q_{i,j} = d_i - d_j$ *is satisfied for root-to-sink delay* d_i *and* d_j *of any sink pair* $v_i, v_j \in V$ *and the maximum load constraint* C_{max} *is met for every buffer and the driver.*

Please note that the minimum required number of skew specifications for n nodes is $n-1$. The other specifications can be derived from the $n-1$ specifications. If more than $n-1$ skew specifications are there, it must be ensured that they are coherent with each other. The skew specifications can also be expressed through root-to-sink delay-target t_i for each sink $v_i \in V$, as long as $q_{i,j} = t_i - t_j \forall v_i, v_j \in V$ is satisfied. In reality, it does not matter whether or not the delay d_i of sink v_i in a clock tree is equal to its delay-target t_i. The skew specifications can be satisfied whenever we can find a single constant C such that $t_i = d_i + C$ is true for every sink $v_i \in V$. The concept of delay-target is employed for the convenience of computation and description. The zero skew requirement can be obtained by letting $t_1 = t_2 = ... = t_n$.

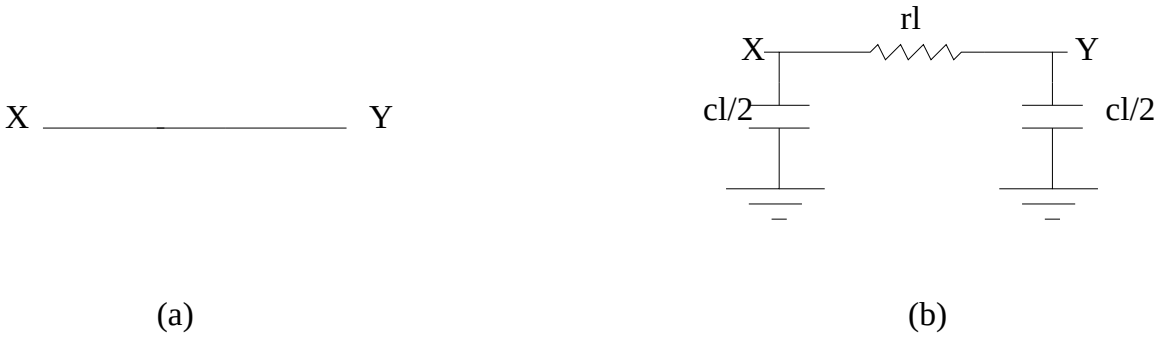


Fig. 5. Interconnect model.

A. Models Used

To calculate the delays, we replace a wire of length l by the RC equivalent circuit shown in Figure 5 using a Pi equivalent. Here r and l are the resistance and capacitance per unit length. A buffer is replaced by the equivalent circuit shown in Figure 6 where input capacitance C_b, intrinsic delay t_b and output resistance R_b are the buffer parameters. Figure 7 shows how a RC tree is created using the buffer and interconnect model.

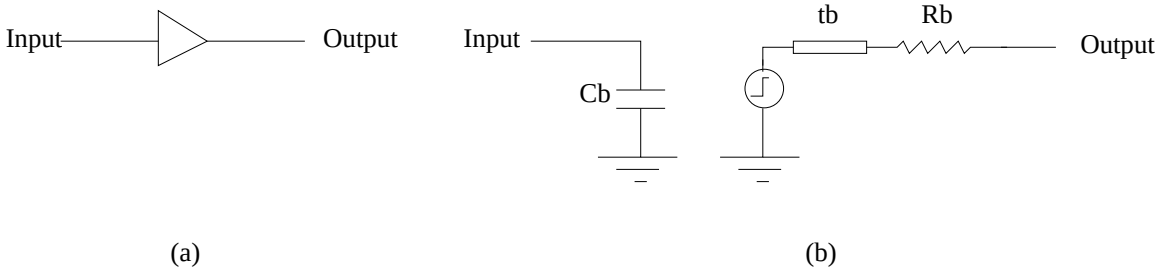


Fig. 6. Buffer model.

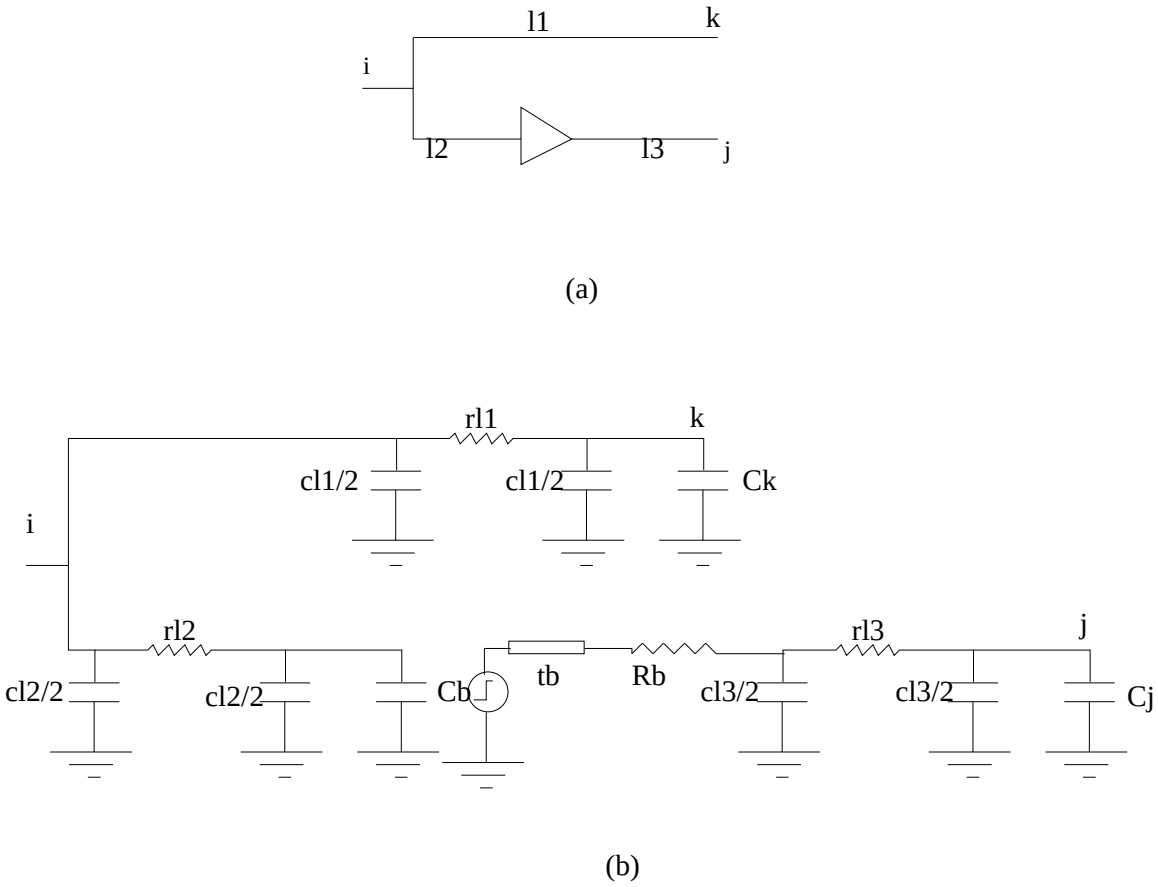


Fig. 7. Equivalent RC circuit using the interconnect model and the buffer model.

B. Delay Balancing

Now we generalize the concept of delay-target to include subtrees. Let T_i denote a subtree rooted at node v_i. This subtree can be characterized by delay-target t_i and downstream capacitance C_i at its root v_i. If v_i is a sink node, its delay-target t_i is given. If v_i is a merging node, its delay-target t_i can be computed recursively as follows. If we merge subtree T_j and T_k at merging node v_i as shown in Figure 8(a), let the wirelength from v_i to v_j and v_k be $l_{i,j}$ and $l_{i,k}$, respectively. Then the delay

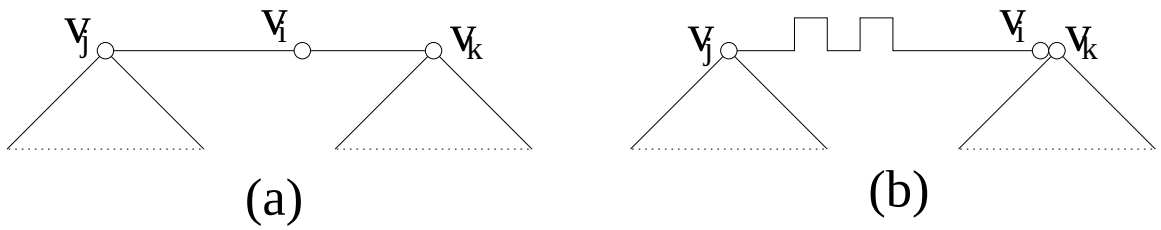


Fig. 8. Examples of merging subtrees without wire snaking in (a) and with wire snaking when delay-target t_j at v_j is significantly greater than delay-target t_k at v_k in (b).

from v_i to v_j and v_k are:

$$d_{i,j} = \frac{1}{2} r c l_{i,j}^2 + r l_{i,j} C_j \tag{2.1}$$
$$d_{i,k} = \frac{1}{2} r c l_{i,k}^2 + r l_{i,k} C_k$$

where r and c are wire resistance and capacitance per unit length, respectively. In order to meet skew specifications, these delays have to satisfy the following equality:

$$d_{i,j} - d_{i,k} = t_j - t_k \tag{2.2}$$

Then the delay-target t_i can be obtained by rearranging the above equality as

$$t_i = t_j - d_{i,j} = t_k - d_{i,k} \tag{2.3}$$

Since the delay-targets are propagated bottom-up based on the above equation, the skew specifications can be enforced by only considering Equation (2.2) without checking delays at sink/leaf nodes. The downstream capacitance C_i can be obtained directly as $C_i = C_j + C_k + c l_{i,j} + c l_{i,k}$.

The minimum feasible wirelength for the merging is the Manhattan distance $l_{j,k}$ between v_j and v_k. The wirelength from v_i to v_j and v_k need to satisfy $l_{j,k} = l_{i,j} + l_{i,k}$. When there is great difference between delay-targets, for example, when t_j is much greater than t_k, we have to let $l_{i,k} = 0$ and let $l_{i,j} > l_{j,k}$ to ensure that the constraint

of Equation (2.2) is met. The actual wirelength of $l_{i,j}$ can be obtained by solving the following equation.

$$\frac{1}{2}rcl_{i,j}^2 + rl_{i,j}C_j = t_j - t_k \tag{2.4}$$

The method of using wirelength greater than $l_{j,k}$ is called wire snaking [10] which is demonstrated in Figure 8(b).

A given buffer type b is characterized by it input capacitance C_b, intrinsic delay t_b and output resistance R_b. When a buffer is inserted at a node v_i, then the delay target at v_i is reduced by $t_b + R_bC_i$ and the downstream capacitance at v_i becomes C_b. Even though a single buffer type is considered in this work, our method can be extended to handle multiple buffer types.

CHAPTER III

ALGORITHM

A. Review of the Deferred-Merge Embedding (DME) Algorithm

The Deffered-Merge Embedding[12] is the best known layout embedding technique and embeds internal nodes of the given topology G using a two-step process. The bottom-up process builds a tree of line segments, where a line segment represents the loci of possible placements of the internal nodes satisfying the skew criterion. This is followed by a top-down process, which resolves the exact locations of all internal nodes in the clock-tree.

1. Bottom-Up Phase: Construction of the Tree of Merging Segments

The abstract tree topology G for the set of sinks is given according to which we need to build the tree of merging segments. Each node v is associated with a merging segment, which depends upon the children of the node v. That is why the construction of the tree of merging segments is a bottom-up process. A length is also assigned to each edge in G and this length is retained in the final embedding of the clock-tree.

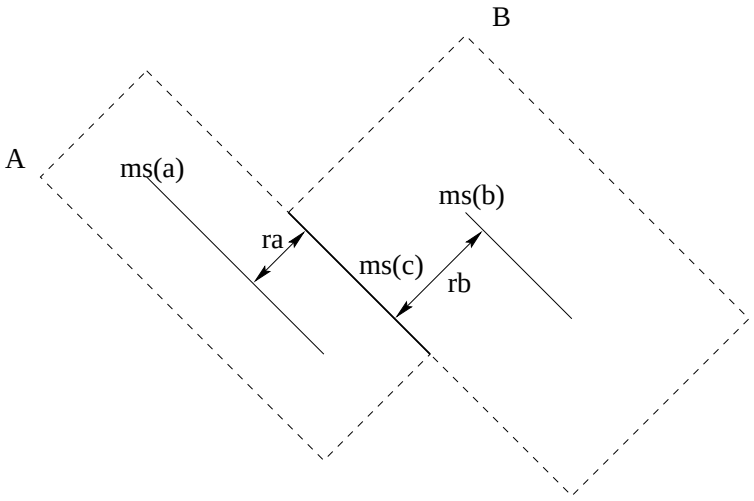
Let a and b be the children of the node v of G. TS_a and TS_b are the sub-trees of merging segments rooted at a and b, respectively. We have to find a placement for v which satisfies the skew criterion for the sub-trees TS_a and TS_b and adds minimum wirelength. Let r_a and r_b be the edge lengths assigned to the segment connecting the node v to the sub-trees TS_a and TS_b respectively. The merging cost is defined as the sum of the edge lengths - $r_a + r_b$. As the delay is a monotone increasing function of the wirelength, the optimal wirelength assignment is unique.

A *Manhattan arc* is defined as a +45 or -45 degrees line segment from the wiring

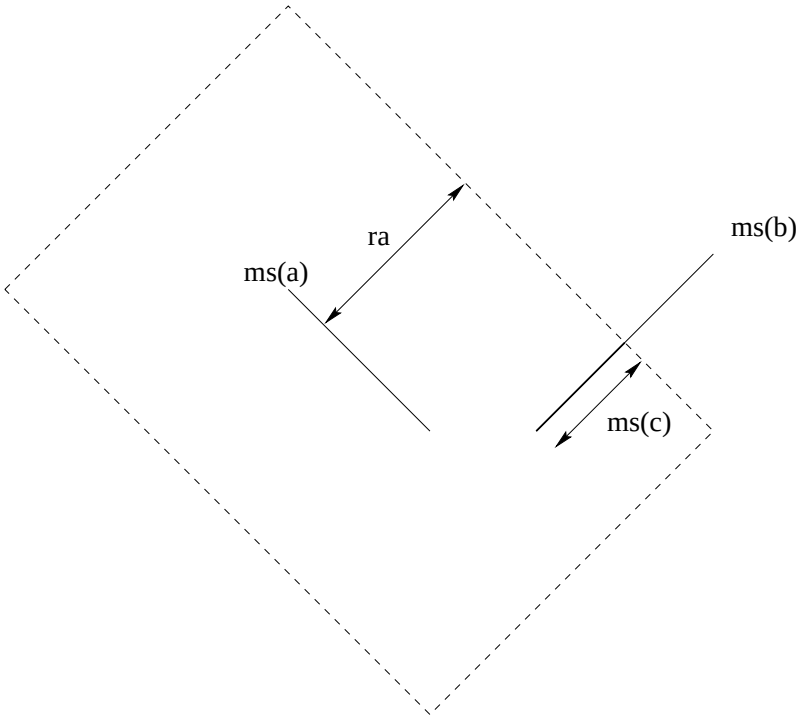
directions. The loci of the points which are at a fixed distance from the *Manhattan arc* is called a *tilted rectangular region*, or *TRR*. The *Manhattan arc* at the center of the *TRR* is called the *core* and the shortest distance between the *core* and *TRR* is the radius. While constructing the merging segment for the node v, we find the *TRRs* of the children a and b. Here we should be aware that the two children of the node v can be both sinks or one sink, one merging segment or both merging segments. In each of these cases, it is proved that the resulting merging segment will also be a *Manhattan arc*[12].

Let us consider two important scenarios of the construction of the merging segment. The first scenario has the skew criterion specified such that wiresnaking is not required to balance the delay. Hence the merging cost at the node v in this case is the minimum distance between *ms(a)* and *ms(b)* which is equal to $r_a + r_b$. Since more than one location can satisfy the skew criterion, the merging segment of the node v is found by first drawing the *TRRa* and *TRRb* using *ms(a)* and *ms(b)* as the *core* and r_a and r_b as the radius and then, calculating *ms(v)* as the intersection of *TRRa* and *TRRb*. Figure 9(a) illustrates the algorithm for the case where the merging cost is equal to the minimum distance between the merging segments of the nodes a and b, and wiresnaking is not required.

The second scenario has skew criterion specified such that wiresnaking is required to balance the delay. This makes one of the edge lengths as 0 and the other equal to the merging cost. The merging cost is higher than the minimum distance between *ms(a)* and *ms(b)*. Again we can have multiple locations satisfying the skew criterion and the merging segment is found as shown in Figure 9(b). Note that in this case only one *TRR* is drawn to find the merging segment.



(a)



(b)

Fig. 9. Construction of merging segment with: (a) no wiresnaking (b) wiresnaking.

The merging segments are found for each of the node of G and edge lengths are stored. The details of the construction of the tree of merging segments is shown in Figure 10. One example of the bottom-up process of construction of the merging segments is shown in Figure 11.

Procedure: $BuildTreeOfSegments(\mathcal{G}, \mathcal{V})$
Input: Topology $\mathcal{G}$; set of sink locations $\mathcal{V}$ **Output:** Tree of merging segments T' containing ms(v) for each node v in $\mathcal{G}$ and edge length r_v for each v$\neq$ *clocksource*
1. **for each v in $\mathcal{G}$ (bottom-up order)** 2. **if v is a sink node and $pl(v)$ is its location or placement** 3. $ms(v) \leftarrow pl(v)$ 4. **else** 5. Let a and b be the children of v 6. Calculate edge lengths r_a and r_b. Create TRRs TRR_a and TRR_b using the edge lengths 7. $ms(v) \leftarrow TRR_a \cap TRR_b$ 8. **endif**

Fig. 10. Construction of the tree of segments.

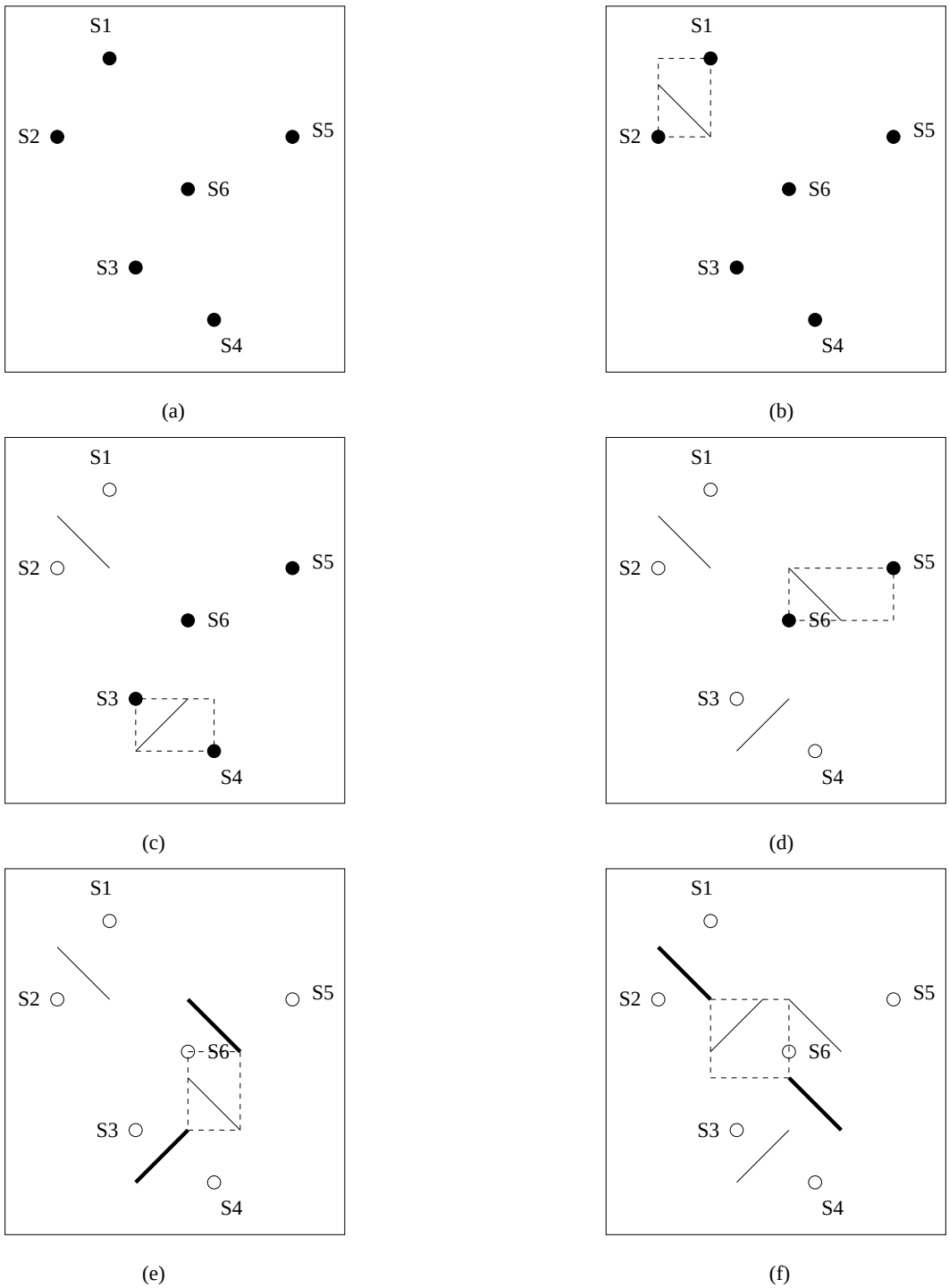


Fig. 11. An example of the bottom-up construction of the merging segment tree.

2. Top-Down Phase: Embedding of Nodes

Once the tree of merging segments is created and the edge lengths are fixed, the embedding of the internal nodes in the clock-tree is done in a top-down process. If the node v is the root node, then select *pl(v)* to be the point on *ms(v)*, which is closest to the clock-source.Otherwise, choose *pl(v)* to be the point which is closest to the parent of the node v. The distance between *pl(v)* and the parent of the node v will always be less than the edge length stored during the bottom-up construction of the merging segments. TRR of the parent, TRR_p is drawn using the edge length as the radius and *pl(v)* is found as any point from $ms(v) \cap TRR_p$. The details of the procedure $FindExactPlacement$ is shown in Figure 12 and the possible placements of the node v given the placement of the parent p is shown in Figure 13.The continuation of the example used in the bottom-up process of construction of the merging segments is shown in Figure 14, which gives the physical clock-tree.

Procedure: $FindExactPlacement(\mathcal{TS})$
Input: Tree of segments $\mathcal{TS}$ containing ms(v) and r_v for each node v. **Output:** Zero skew tree
1. **for each v in G (top-down order)** 2. **if v is the root** 3. Choose *pl(v)* on *ms(v)*,which is nearest to clock source 4. **else** 5. Let p be the parent node of v 6. Construct TRR_p using the edge length r_p 7. Choose any $pl(v)$ on $ms(v) \cap TRR_p$ 8. **endif**

Fig. 12. Fixing the exact location of internal nodes in the zero skew clock tree.

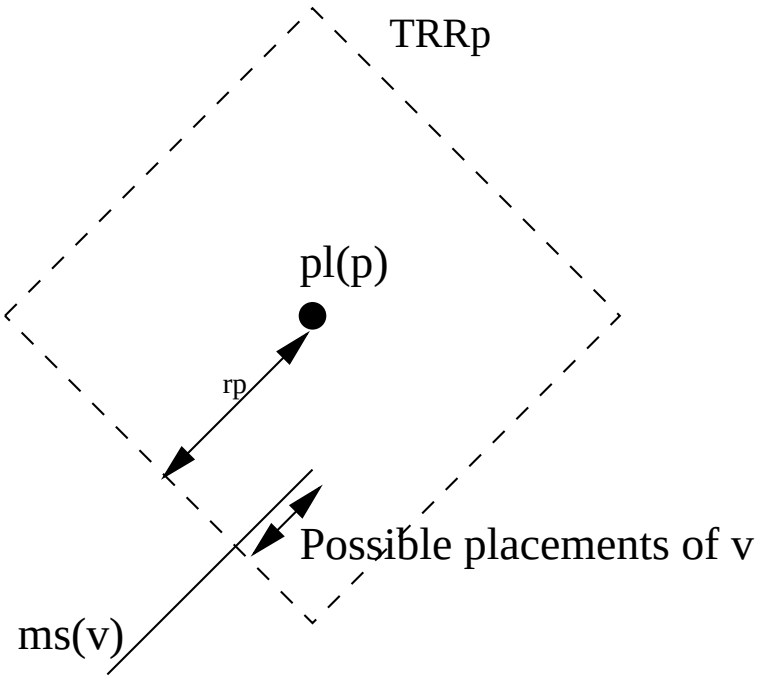


Fig. 13. Fix the location of v given the placement of the parent p.

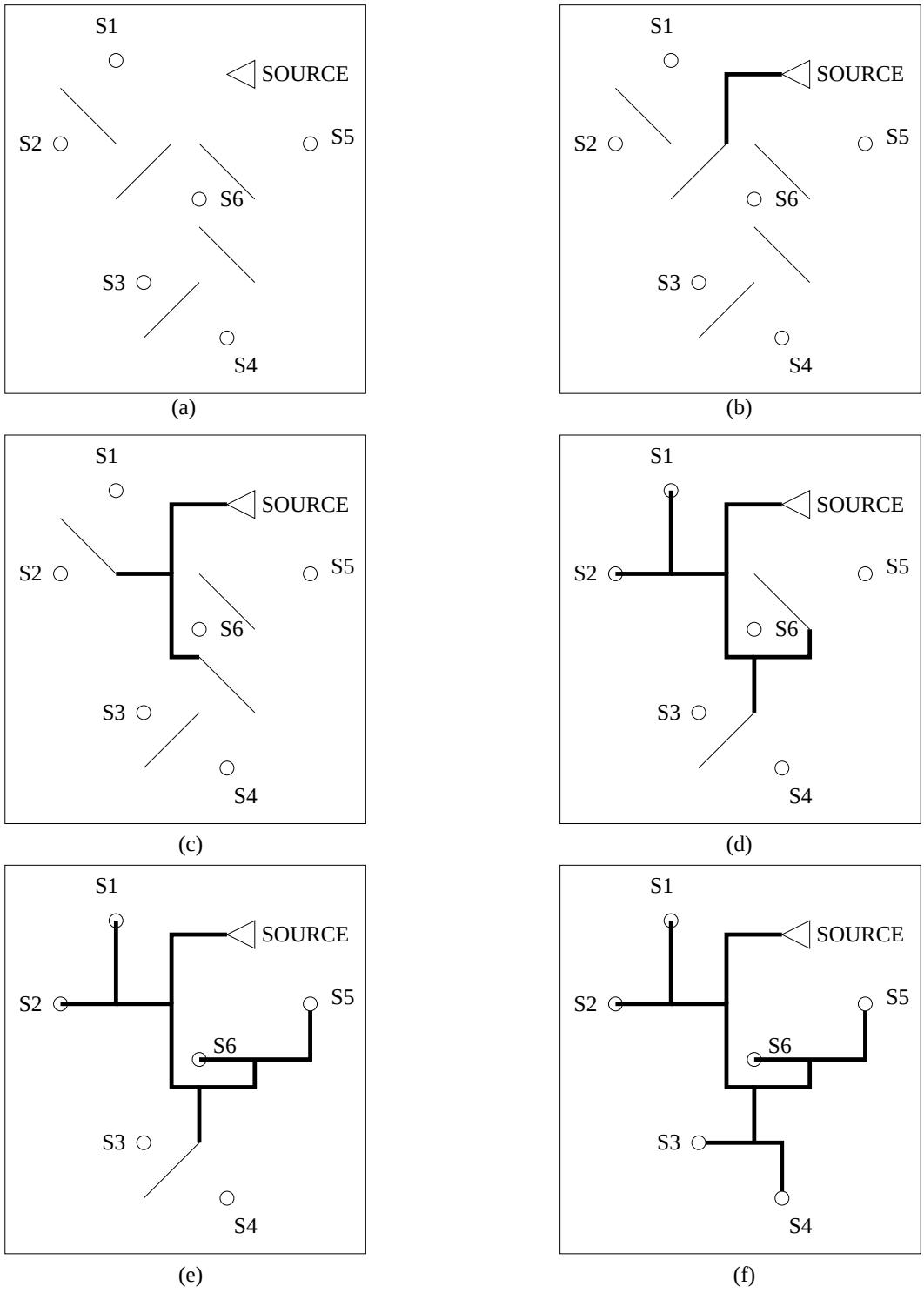


Fig. 14. An example of the top-down embedding of internal nodes.

B. The Merging Scheme

The top-level framework of our algorithm[26] is similar to Edahiro's NS algorithm[13], however, we propose a merging scheme and a buffering method that are designed particularly for prescribed non-zero skew clock routing. The proposed merging scheme and buffering method will be described in detail as follows.

Most of previous merging schemes [9, 13] choose the subtree pair with the minimum distance between their roots and merge them first. Their attentions are only at subtree spatial proximities, since delay-targets are identical for all sinks in zero skew routing. It is shown in the previous section that great difference between delay-targets may cause wire snakings, thus traditional merging schemes tend to result in excessive wirelength because of their neglection of the delay-target differences. We demonstrate this problem through the example in Figure 1. Assume that the given delay-targets are quite different from each other and they follow the inequality $t_1 < t_2 << t_3 < t_4$, especially t_3 and t_4 are much greater than t_1 and t_2. We merge T_1 with T_2 first, since their distance is the smallest among all sink pairs. Because t_2 is significantly greater than t_1, it is quite likely that a wire snaking occurs when we merge T_1 with T_2 at node v_5 as shown in Figure 1(b). Similarly, T_3 is merged with T_4 at node v_6. Since t_3 and t_4 are much greater than t_1 and t_2, it is quite possible that t_6 is much greater than t_5 and another wire snaking results from merging subtree T_5 with T_6 at node v_7.

Since wire snaking is more likely to happen when the difference of delay-targets between two subtrees is large, it can be reduced if we choose a merging order that can reduce the delay-target differences among all subtrees. According to Equation (2.3), the delay-target of the newly created subtree is always smaller than the delay-targets of the two subtrees it is merged from. Thus, if we choose to merge the subtree with the maximum delay-target first, the overall delay-target differences among subtrees

will be reduced. We can analogize the set of subtrees as a group of runners. We let the runner lagging behind run first so that he/she is closer to runners ahead of him/her. According to Equation (2.1), if C_j is much greater than C_k, it is easier to achieve great $d_{i,j} - d_{i,k}$ without wire snaking. When the maximum delay-target subtree is merged first, the newly created subtree from this merging has not only a smaller delay-target but also a greater load capacitance that makes the matching to other small delay-target subtrees easier. Therefore, the maximum delay-target ordered merging can reduce the chance of wire snaking by decreasing delay-target imbalance and increasing load imbalance that is coherent with the delay-target imbalance.

We further illustrate the advantage of this maximum delay-target ordered merging through the example in Figure 1. In Figure 1(d), we first merge T_3 with T_4 to obtain subtree T_6 rooted at v_6, as v_4 has the maximum delay-target. Since t_4 and t_3 are much greater than t_2 and t_1, it is very likely that t_6 is still greater than t_1 and t_2. Next, we merge T_6 with T_2 at node v_8 and denote this merging as $T_6 + T_2 \rightsquigarrow v_8$. We can compare this merging with $T_6 + T_5 \rightsquigarrow v_7$ in Figure 1(b), since both mergings start from v_6. On one hand, there is less imbalance on delay-targets for merging $T_6 + T_2 \rightsquigarrow v_8$ since $t_6 - t_2 < t_6 - t_5$. On the other hand, as $C_2 < C_5$, the merging $T_6 + T_2 \rightsquigarrow v_8$ has greater imbalance on load capacitance which makes it easier to achieve imbalanced delay-targets without wire snaking. If we compare the merging $T_1 + T_8 \rightsquigarrow v_9$ in Figure 1(d) and the merging $T_1 + T_2 \rightsquigarrow v_5$ in Figure 1(b), same conclusion can be obtained. Therefore, the maximum delay-target first merging indeed reduces the chance of wire snaking.

Besides the maximum delay-target criterion, there is another major difference between our merging scheme and previous works. Previous works such as [13] evaluate every *pair* of subtrees and choose a pair according to the minimum distance criterion. Our maximum delay-target criterion only selects a *single* subtree instead of a pair

at once, and we will apply another criterion to choose another subtree (we call it *companion subtree*) to be merged with the maximum delay-target subtree. If we pick the subtree which is closest to the maximum delay-target tree as a companion, then the neglection on the delay-target difference between them may again result in wire snakings. If we pick the subtree with the closest delay-target, these two subtrees may be far apart from each other and the merging may cause large wirelength too. Therefore, a subtree needs to be merged to another subtree that is not only nearby but also with similar delay-target. In other words, we need to play in a three-dimensional space of $(x, y, delay\_target)$. We introduce a *merging cost* to include the concern on distance and delay-targets in a unified form. This merging cost is simply the wirelength needed for the merging to satisfy the delay-target constraint (2.2). Therefore, the merging cost is same as the Manhattan distance between the roots of two subtrees if there is no wire snaking. Otherwise, the merging cost is obtained through solving Equation(2.4) to include the extra wirelength due to wire snaking. Therefore we choose the companion subtree, which will lead to the minimum *merging cost*.

The algorithm description for this merging scheme is given in Figure 15. In fact, the proposed merging scheme is effective on reducing wirelength for zero skew routing as well. Even though every sink initially has the same delay-target in zero skew routing, delay-targets of the subtrees after merging are quite likely to be different from each other. If we treat the post-merging subtrees as pseudo sinks, the remaining clock routing task is equivalent to a non-zero skew clock routing. The effect of our merging scheme depends on how large the delay-target differences are. The larger the delay-target difference, the more effective our merging scheme is.

Procedure: $FindSubtreesToBeMerged(\mathcal{T})$
Input: A set of subtrees $\mathcal{T}$ **Output:** Two subtrees to be merged
1. $T_i \leftarrow$ subtree with the maximum delay-target in $\mathcal{T}$ 2. $minCost \leftarrow \infty$ 3. For each subtree $T_j \in \mathcal{T} \backslash T_i$ 4. $cost \leftarrow$ merging cost between T_i and T_j 5. If $cost < minCost$ 6. $minCost \leftarrow cost$, $T_k \leftarrow T_j$ 7. Return T_i and T_k

Fig. 15. Algorithm of the merging selection scheme.

C. Buffer Insertion

Buffers are inserted during the process of merging subtrees to accomplish two objectives: (1) enforcing a load capacitance constraint; and (2) reducing wire snakings.

1. Load Constraint

The load capacitance constraint C_{max} specifies the maximum load capacitance which a buffer/driver can drive. Since the output slew rate of a buffer/driver is mainly determined by its load capacitance[27], restraining the load capacitance can virtually keep a signal slew rate at proper level. The load capacitance constraint can be satisfied through either dynamic programming style algorithms[22, 28] or a greedy approach[24]. In a dynamic programming algorithm, since a set of candidate solutions are maintained, any candidate solution with violation on the constraint can be simply pruned out. Aimed to a fast and practical solution, this work adopts the greedy approach as in [24].

If we check the load constraint criterion at the root of sub-trees and insert buffers at the root only, then we may have a load constraint violation at some intermediate point on the edges even after buffer insertion. To avoid this, we do not merge those sub-trees whose merging point violates the load constraint but we do insert buffers at the root of such sub-trees[29]. This lowers the node capacitance and prevents the load constraint violation in the future merging of the sub-tree. The merging selection is done again to get the new pair of sub-trees. The details of the procedure *ApplyLoadConstraint* is shown in Figure 16.

Procedure: *ApplyLoadConstraint*($TS1, TS2$)
Input: Two subtrees to be merged **Output:** Two subtrees which do not violate load constraint
1. Let TS be the resultant sub-tree which would be obtained by merging $TS1$, $TS2$
2. **If** downstream capacitance C_{TS} at $TS > C_{max}$
3. Insert buffers at root of sub-trees $TS1$, $TS2$ appropriately.
4. *FindSubtreesToBeMerged(T)*. Let the returned sub-trees be $TS1$, $TS2$.
5. *ApplyLoadConstraint(TS1, TS2)*
6. **else** Return $TS1$, $TS2$

Fig. 16. Algorithm for applying the load constraint.

2. Delay Balancing

Sometimes a buffer may be inserted to reduce wire snaking[29]. A buffer is inserted only if the extra wire capacitance due to the wire snaking is greater than the input capacitance C_b of the buffer. A remarkable wire snaking often happens when there is great imbalance between the delay-targets of the two subtrees to be merged. The delay-target t_j for subtree T_j can be reduced by $R_b C_j + t_b$ through adding buffer at

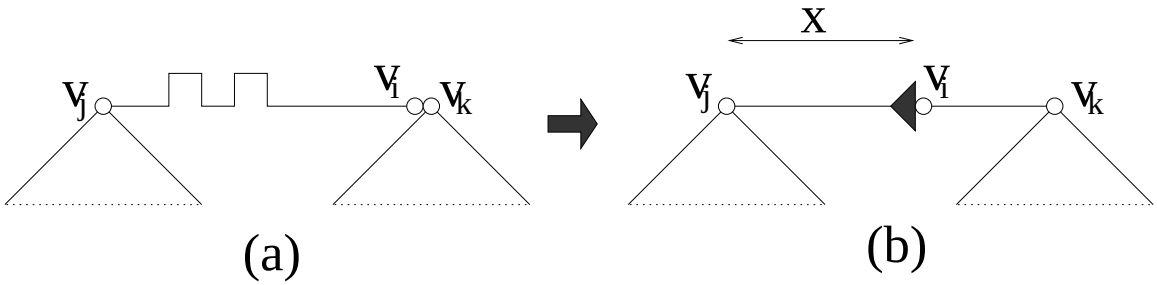


Fig. 17. Buffer insertion to reduce wire snaking. Delay target $t_j > t_k$.

root v_j. Please note this conclusion is contrary to the case in signal routing where buffers are usually employed to reduce delay[30]. For the example in Figure 17, since $t_j > t_k$, a buffer is inserted to drive the branch with T_j as in Figure 17(b).

Next, the buffer location needs to be decided in addition to the merging node v_i location. The work of [24] would simply fix the buffer location at the root v_j. In contrast, we do not restrict the buffer location at v_j so that a larger solution space can be explored. In order to avoid solving two separate location variables simultaneously, we let the buffer and the merging node v_i be at a same location. This simplification does not sacrifice any solution space for a reason that is illustrated by the example in Figure 17(b). In Figure 17(b), the maximum delay d_{max} between v_i and v_j occurs when both the buffer and the merging node v_i are at the location of v_k, i.e., $x = l_{j,k}$. Similarly, the minimum delay d_{min} between v_i and v_j occurs when the buffer and v_i are at the location of v_j, i.e., $x = 0$. By moving the buffer and v_i together between v_j and v_k (varying x in $[0, l_{j,k}]$), any value in $[d_{min}, d_{max}]$ can be obtained for the delay between v_i and v_j. The value of x is decided according to skew specifications.

The buffer insertion for wire snaking reduction may introduce inaccuracy to the *merging cost* in searching the companion subtree, since the merging cost does not count the buffer insertion effect. However, this inaccuracy is limited because the

neglected snaking cost reduction is offset by the extra buffer cost. On the other hand, neglecting the buffer effect makes the merging scheme faster.

D. Complexity

When integrated with the DME embedding as in [13], the merging will be performed $n-1$ times for n clock sinks. The complexity of merging selection is $O(n)$ due to the loop of line 3-6 in Figure 15. For each merging, the computation of buffer location and merging node location takes constant time. Thus, the overall complexity of our buffered clock routing algorithm is $O(n^2)$.

CHAPTER IV

EXPERIMENTS AND RESULTS

A. Experimental Setup

We implemented the proposed buffered clock tree algorithm in C and experiments are performed on a PC with 1.7GHz Pentium 4 microprocessor and 512Mb memory. The benchmark circuits are prim1, prim2 and r1-r5 downloaded from the GSRC Bookshelf ($http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/BST/$). The delay-targets are generated through running the BST [12] code with a global skew bound of $100ps$ and taking the non-zero skew results. The BST code is also downloaded from the GSRC Bookshelf.

ip_sample is a small testcase with prescribed-skew specifications taken from a real chip of IBM. The details of *ip_sample* testcase is shown in Appendix A.

B. Unbuffered Prescribed-skew Clock-tree

For non-zero skew targets, we also implemented and extended the NS algorithm [13], one of the best zero skew routing algorithms, for comparison. Since our merging scheme includes two major components: (i) the maximum delay-target subtree first and (ii) choosing companion subtree according to the minimum merging cost, we tried them separately to observe each individual effect[26]. Therefore, we compared the following four clock routing algorithms with different merging schemes:

- NS: The Nearest-neighbor Selection algorithm in[13] using the minimum distance merging. The DME implementation is extended such that non-zero skew can be achieved. Figure 18 shows the resultant clock-tree generated, along with the node numbers, by this algorithm for the testcase *ip_sample*.

- MIC: The MInimum merging-Cost merging which is very similar to NS except that the merging selection is based on the merging cost between two subtrees instead of the distance between them. Figure 19 shows the resultant clock-tree generated, along with the node numbers, by this algorithm for the testcase *ip_sample*.

- MAT: Choose the MAximum delay-Target subtree first and find its companion subtree that is its nearest neighbor(with minimum distance). Figure 20 shows the resultant clock-tree generated, along with the node numbers, by this algorithm for the testcase *ip_sample*.

- MAT-MIC: This is the complete version of our proposed merging scheme, which is a combination of previous two techniques. First choose the maximum delay-target subtree and then find its companion subtree which results in the minimum merging cost between them. Figure 21 shows the resultant clock-tree generated, along with the node numbers, by this algorithm for the testcase *ip_sample*.

The experimental results for non-zero skew targets are shown in Table I. Since these clock routing algorithms all deliver the same prescribed non-zero skews, we only report the total wirelength here. The percentage reduction on wirelength with respect to NS are listed in column 5. We can see that either the maximum delay-target or the minimum merging-cost technique itself can make significant improvement on wirelength over the naive extension from previous zero skew routing NS. The improvement from the merging-cost based criterion alone is from 24% to 51%. The effect from the maximum delay-target ordering is more remarkable and shows 39%-53% wirelength reduction. The combination of these two techniques, which is our proposed merging scheme, yields wirelength improvement of 53%-61%. The CPU time for each routing algorithm are shown in the rightmost column of Table I.

Table I. Comparisons of clock routing with different merging schemes under non-zero skew targets. The wirelength reduction is with respect to the wirelength from NS of [13].

Testcase	#sinks	Merging	Wirelength	Wire reduction	CPU(sec)
prim1	269	NS	271746	-	2
		MIC	152636	44.0%	5
		MAT	161248	40.6%	1
		MAT-MIC	109056	59.9%	1
prim2	603	NS	645248	-	22
		MIC	490895	23.9%	41
		MAT	314455	51.3%	1
		MAT-MIC	259751	59.7%	1
r1	267	NS	2477975	-	1
		MIC	1515972	38.8%	4
		MAT	1491833	39.8%	1
		MAT-MIC	1160145	53.18%	1
r2	598	NS	4935718	-	20
		MIC	2936488	40.5%	44
		MAT	2656833	46.1%	1
		MAT-MIC	2243723	54.5%	1
r3	862	NS	6847146	-	56
		MIC	3927066	42.7%	118
		MAT	3242982	52.6%	1
		MAT-MIC	2851177	58.4%	1
r4	1903	NS	15123351	-	462
		MIC	7358685	51.3%	1250
		MAT	7283887	51.9%	3
		MAT-MIC	5819055	61.5%	4
r5	3101	NS	23013115	-	2572
		MIC	11076659	51.9%	6234
		MAT	10695010	53.5%	8
		MAT-MIC	8810532	61.7%	11

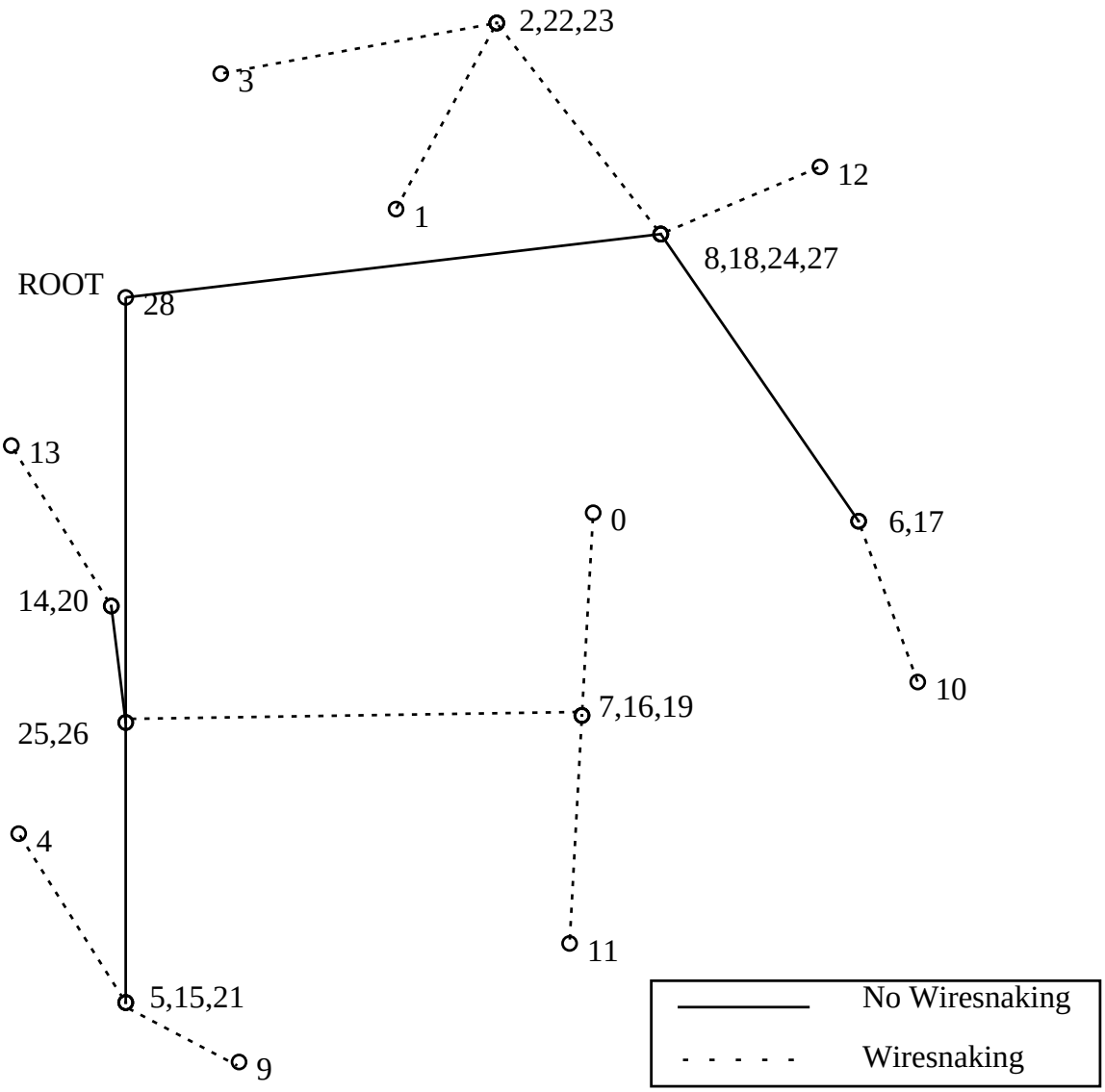


Fig. 18. Clock tree obtained by NS.

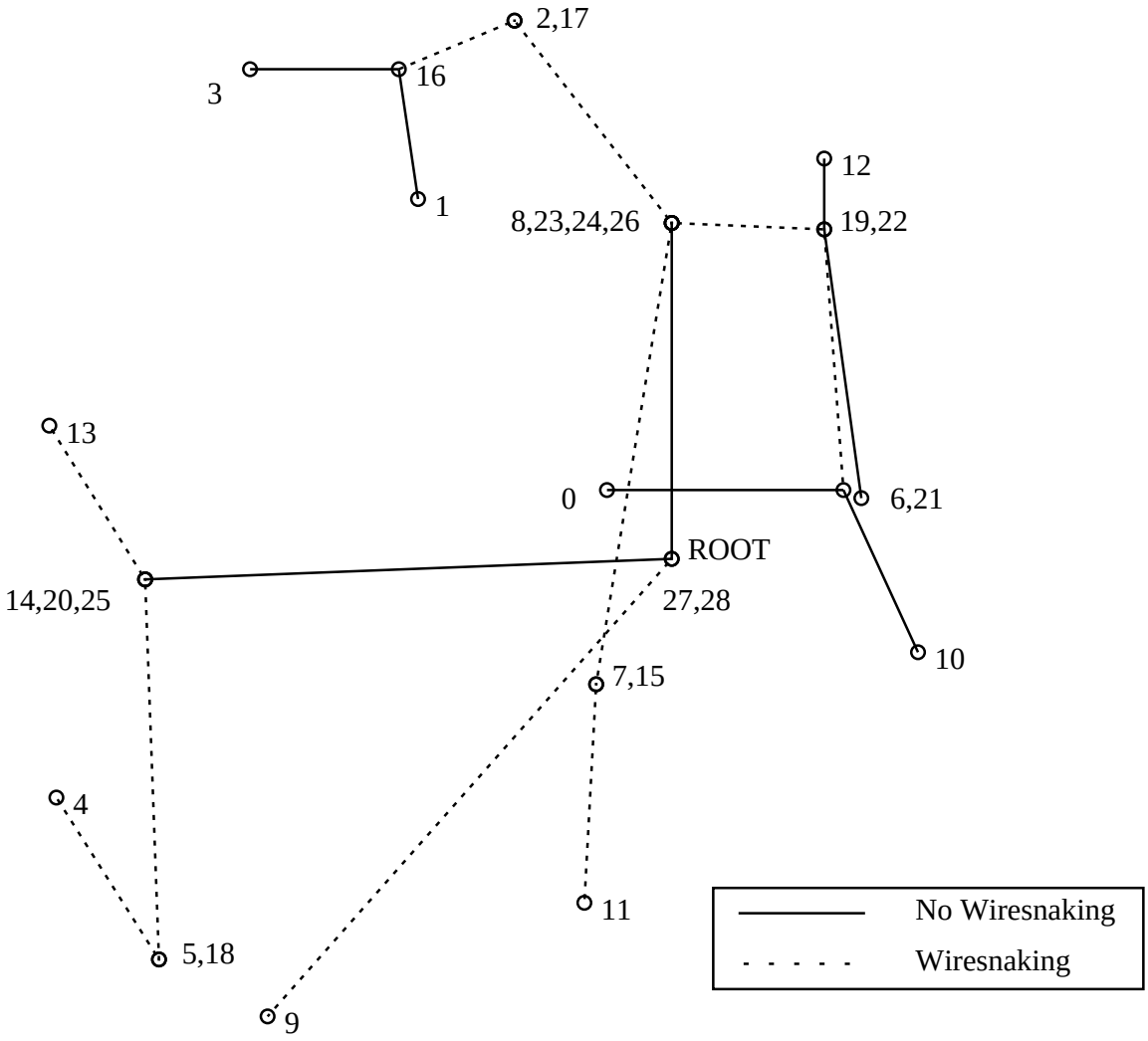


Fig. 19. Clock tree obtained by minimum merging cost based algorithm - MIC.

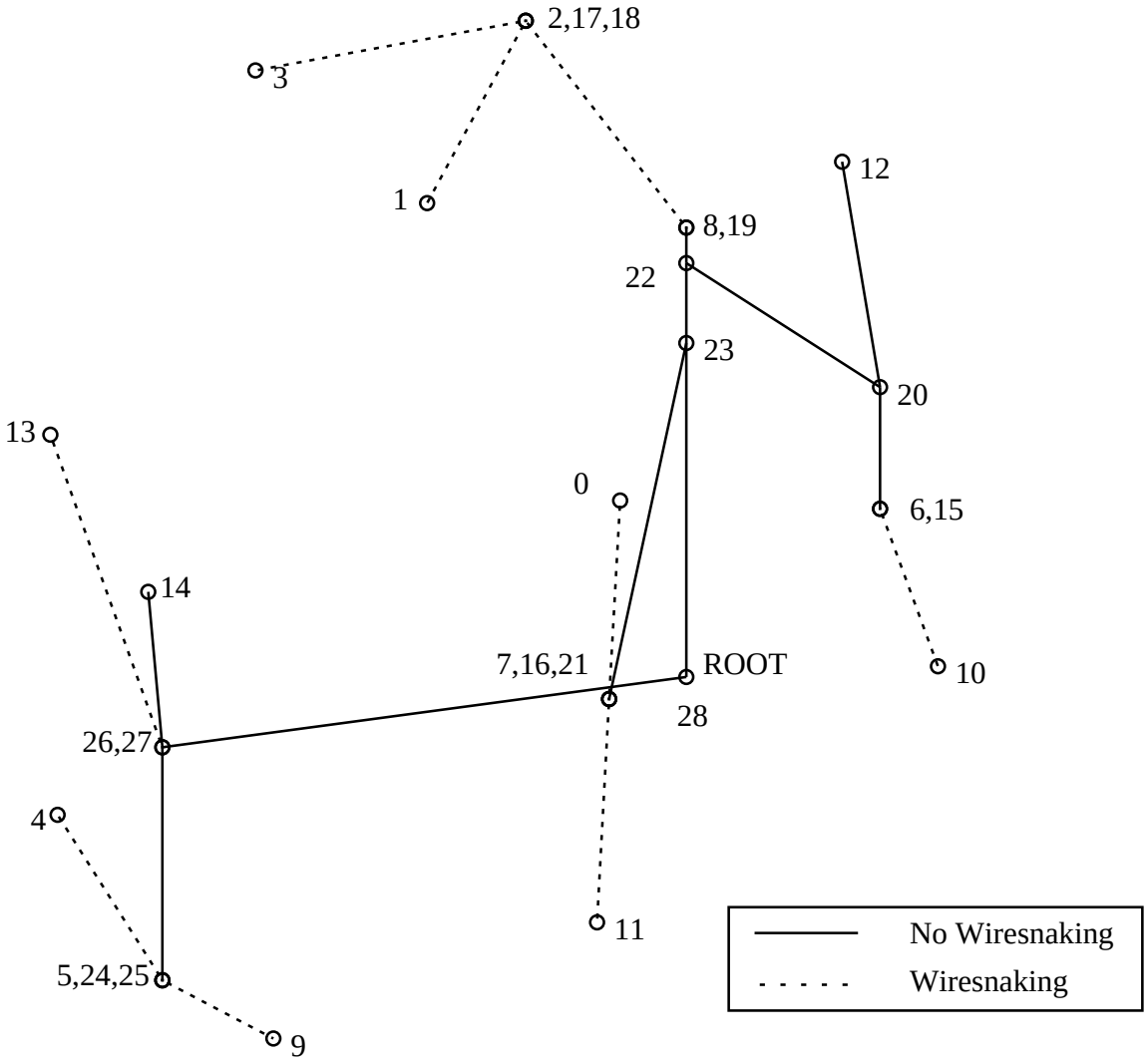


Fig. 20. Clock tree obtained by maximum delay target based ordering algorithm - MAT.

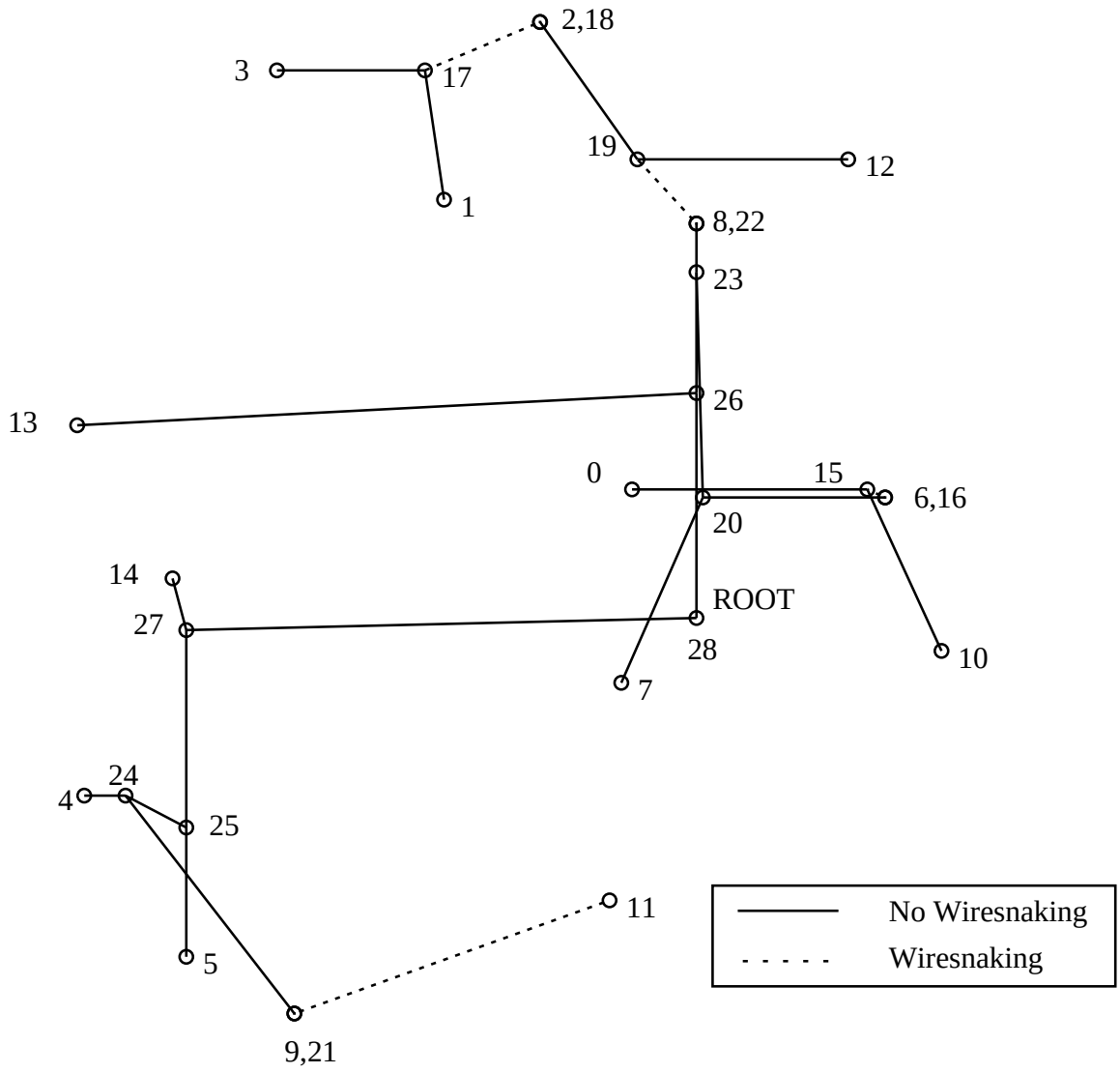


Fig. 21. Clock tree obtained by maximum delay target and minimum merging cost based ordering algorithm - MAT-MIC.

C. Buffered Prescribed-skew Clock-tree

For comparison, the NS(Nearest-neighbor Selection) algorithm proposed in[13] is extended for non-zero skew targets and combined with the same buffering scheme as ours[29].

The experimental results are shown in Table II. Since both algorithms deliver the same prescribed non-zero skews, we only report the resource consumptions including total wirelength, the number of buffers inserted and the total wire and buffer capacitance. The overall improvements are listed in the last row. For all three resource consumption metrics, our algorithm results in huge improvement. The CPU time are shown in the rightmost column of Table II. Note that our buffered clock routing is not only effective but also fast for practical applications. Figure 22 and Figure 23 show the resultant buffered clock-tree generated by the Nearest-neighbor Selection algorithm and MAT-MIC algorithm, respectively, for the testcase *ip_sample*.

Table II. Comparison of our buffered clock tree routing and an extension to the NS algorithm [13].

Testcase	#sinks	Algorithm	Wirelength	#bufs	Wire cap + Buf cap(pF)	CPU(sec)
prim1	269	NS+	718533	59	20.8	1
		Ours	181982	34	5.7	1
prim2	603	NS+	2030400	139	58.1	11
		Ours	480017	73	14.7	1
r1	267	NS+	2917614	49	59.5	1
		Ours	1293616	13	26.2	1
r2	598	NS+	5881313	93	119.8	13
		Ours	2541324	25	51.4	1
r3	862	NS+	7287088	112	148.4	42
		Ours	3265058	28	66.0	1
r4	1903	NS+	16276930	474	331.1	474
		Ours	6659865	62	134.6	3
r5	3101	NS+	24933092	1968	507.7	1968
		Ours	9860004	99	199.5	10
Overall improvement			59.6%	69.0%	60.0%	99.3%

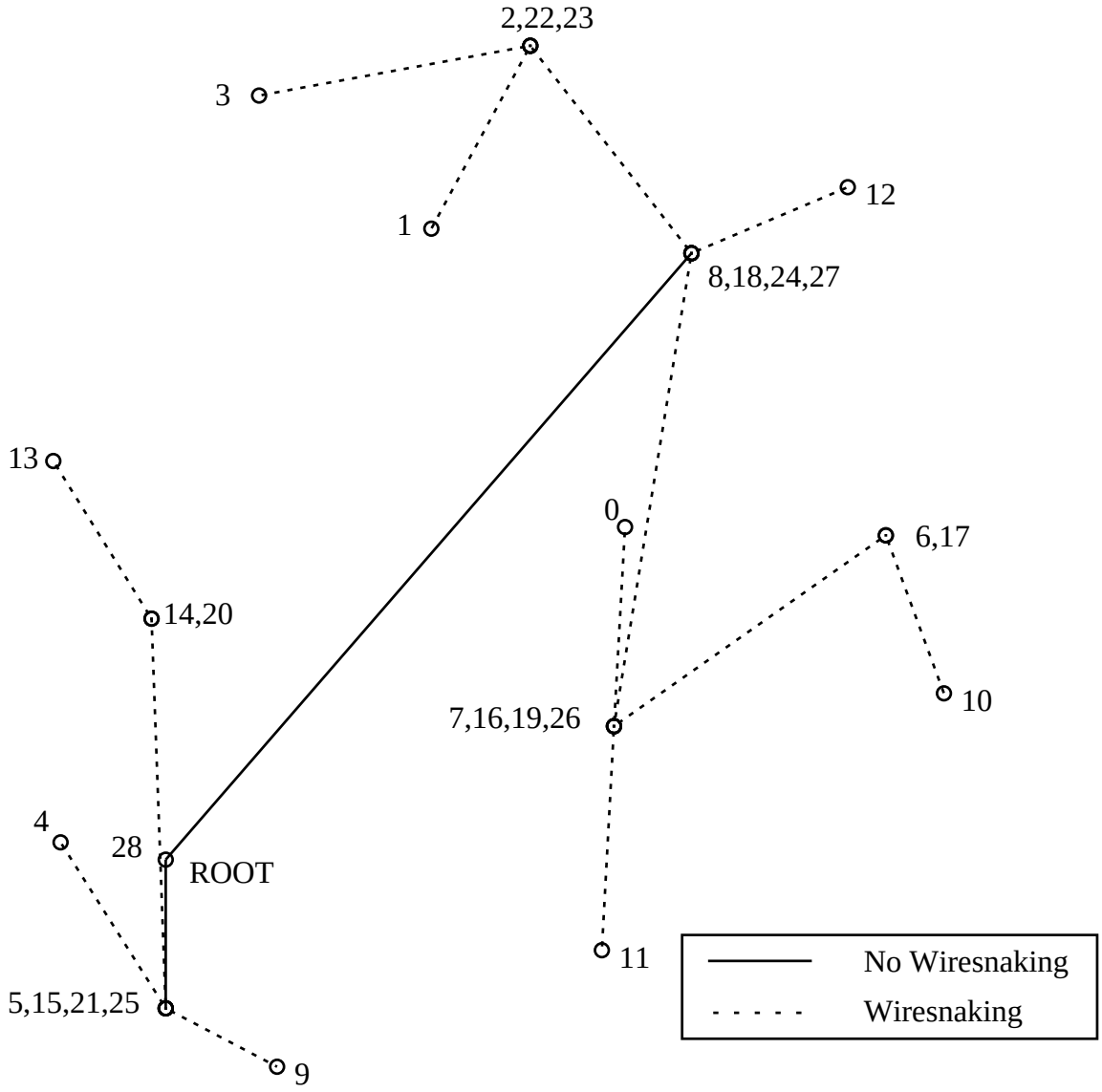


Fig. 22. Buffered clock tree obtained by extended zero skew tree along with load constraint.

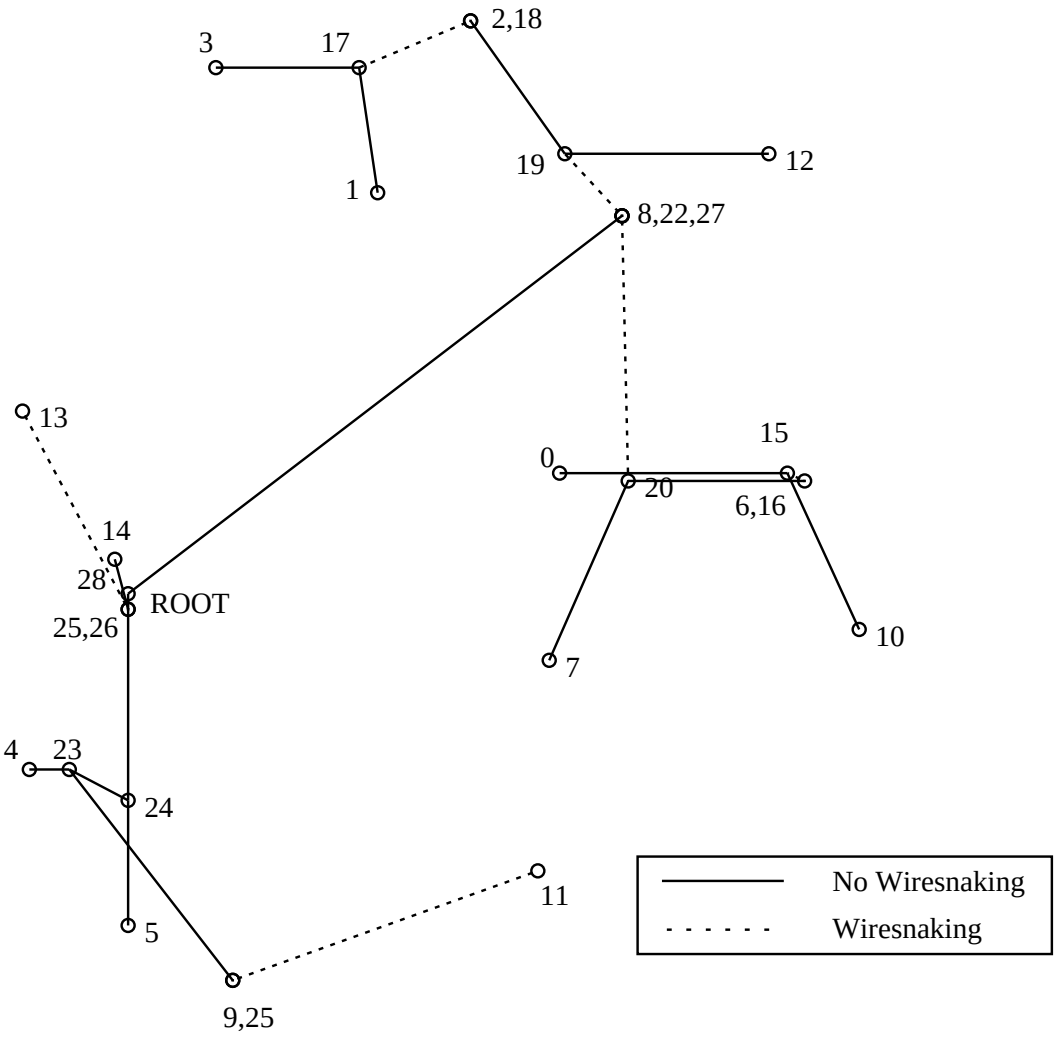


Fig. 23. Buffered clock tree obtained by maximum delay target and minimum merging cost based ordering along with load constraint.

D. Zero-skew Clock-tree

We also compared our algorithm[31] for zero skew routings with algorithms in [13]. In [13], besides the basic version NS, a speed-up version CL is proposed together with a local exhaustive search based post processing algorithm which is notated as I6. The speed-up version CL results in moderately less wirelength than the basic version NS in practice. The post processing step can be performed iteratively to further reduce wirelength for a given clock tree. We compared zero skew clock routing wirelength from our algorithm with the CL in Table III. Since the delay-target differences among subtrees in zero skew routing are normally not large, the MIC technique is rarely useful. However, the MAT merging scheme always yields less wirelength than CL as shown in Table III. The wirelength data of CL are from [13]. The CPU time is not provided in [13]. In Table III, the ratio of wirelength with respect to the CL+I6 flow for each test is listed in parentheses. Our MAT algorithm results in slightly greater wirelength than CL+I6, though never more than 4%. It is expected that a MAT+I6 flow may make the gap even smaller or diminished. For single pass constructive algorithms, our MAT algorithm always provides better solution than CL. Moreover, without Delaunay triangulations, the implementation of MAT is easier than CL.

Table III. Wirelength from our MAT algorithm and CL algorithm in [13]. The number in each parentheses is the ratio with respect to wirelength from CL+I6 algorithm in [13].

Testcase	CL(/[CL+I6])		MAT(/[CL+I6])	
prim1	132980	(1.03)	131716	(1.02)
prim2	334107	(1.10)	312504	(1.03)
r1	1421307	(1.13)	1289459	(1.03)
r2	2627494	(1.06)	2587492	(1.04)
r3	3550494	(1.11)	3282424	(1.03)
r4	6794605	(1.05)	6643357	(1.02)
r5	10195581	(1.05)	9839246	(1.01)

CHAPTER V

CONCLUSION

Even though traditional zero skew routing methods can be applied to achieve non-zero skews, they may bring huge wire and buffer area overhead as the difference among sink delay-targets are ignored in their merging schemes. We propose a merging scheme based on the maximum delay-target and minimum merging cost based ordering and integrate it with deferred-merge embedding technique. This merging scheme helps in avoiding the wiresnaking by increasing the node capacitance and decreasing the delay-target difference between the sinks. Buffers are also inserted to apply the load constraint and bring the slew rate under acceptable range. Buffers also help in balancing the delay of the sinks with huge delay-target difference and results in less wiresnakings. Experimental results on benchmark circuits show that our buffered clock routing algorithm is effective on minimizing wire and buffer area for non-zero skew specifications. The proposed algorithm is effective in case of zero-skew clock-trees and performs even better if the number of sinks increases.

REFERENCES

[1] J. P. Fishburn. "Clock skew optimization," *IEEE Trans. on Computers*, vol. 39, pp. 945–951, July 1990.

[2] Y. Liu, S. R. Nassif, L. T. Pileggi, and A. J. Strojwas. "Impact of interconnect variations on the clock skew of a gigahertz microprocessor," In *Proc. Design Automation Conference*, Los Angeles, CA, June 2000, pp. 168–171.

[3] S. Zanella, A. Nardi, A. Neviani, M. Quarantelli, S. Saxena, and C. Guardiani. "Analysis of the impact of process variations on clock skew," *IEEE Trans. on Semiconductor Manufacturing*, vol. 13(4), pp. 401–407, Nov. 2000.

[4] R. B. Deokar and S. S. Sapatnekar. "A graph-theoretic approach to clock skew optimization," In *Proc. Int. Symposium on Circuits and Systems*, London, May 1994, pp. 1.407–1.410.

[5] H. B. Bakoglu, *Circuits, interconnections and packaging for VLSI.* Reading, MA: Addison-Wesley, 1990.

[6] W.-C. D. Lam, C.-K. Koh, and C.-W. A. Tsao. "Power supply noise suppression via clock skew scheduling," In *Proc. IEEE Int. Symposium on Quality Electronic Design*, San Jose, CA, March 2002, pp. 355–360.

[7] I. S. Kourtev and E. G. Friedman. "Clock skew scheduling for improved reliability via quadratic programming," In *Proc. Int. Conf. on Computer Aided Design*, San Jose, CA, Nov. 1999, pp. 239–243.

[8] M. A. B. Jackson, A. Srinivasan, and E. S. Kuh. "Clock routing for high-performance ICs," In *Proc. Design Automation Conference*, Orlando, FL, July 1990, pp. 573–579.

[9] A. B. Kahng, J. Cong, and G. Robins. "High-performance clock routing based on recursive geometric matching," In *Proc. Design Automation Conference*, San Francisco, CA, June 1991, pp. 322–327.

[10] R.-S. Tsay. "Exact zero skew," In *Proc. Int. Conf. on Computer Aided Design*, Santa Clara, CA, Nov. 1991, pp. 336–339.

[11] 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. on Circuits and Systems - Analog and Digital Signal Processing*, vol. 39(11), pp. 799–814, Nov. 1992.

[12] J. Cong, A. B. Kahng, C.-K. Koh, and C.-W. A. Tsao. "Bounded-skew clock and Steiner routing," *ACM Trans. on Design Automation of Electronic Systems*, vol. 3(3), pp. 341–388, July 1998.

[13] M. Edahiro. "A clustering-based optimization algorithm in zero-skew routings," In *Proc. Design Automation Conference*, Dallas, TX, June 1993, pp. 612–616.

[14] C.-W. A. Tsao and C.-K. Koh. "UST/DME: a clock tree router for general skew constraints," In *Proc. Int. Conf. on Computer Aided Design*, San Jose, CA, Nov. 2000, pp. 400–405.

[15] J. G. Xi and W. W.-M. Dai. "Useful-skew clock routing with gate sizing for low power design," *Journal of VLSI Signal Processing*, vol. 16(2/3), pp. 163–179, June/July 1997.

[16] S. Pullela, N. Menezes, J. Omar, and L. T. Pillage. "Skew and delay optimization for reliable buffered clock trees," In *Proc. Int. Conf. on Computer Aided Design*, San Jose, CA, Nov. 1993, pp. 556–562.

[17] Y. P. Chen and D. F. Wong. "An algorithm for zero-skew clock tree routing with buffer insertion," In *Proc. European Design and Test Conference*, Paris, France, March 1996, pp. 230–236.

[18] J. G. Xi and W. W.-M. Dai. "Buffer insertion and sizing under process variations for low power clock distribution," In *Proc. Design Automation Conference*, San Francisco, CA, June 1995, pp. 491–496.

[19] M. Edahiro and R. J. Lipton. "Clock buffer placement algorithm for wire-delay-dominated timing model," In *Proc. Great Lake Symposium on VLSI*, Ames, IA, March 1996, pp. 143–147.

[20] A. Vittal and M. Marek-Sadowska. "Power optimal buffered clock tree design," In *Proc. Design Automation Conference*, Las Vegas, NV, June 1996, pp. 230–236.

[21] X. Zeng, D. Zhou, and W. Li. Buffer insertion for clock delay and skew minimization. In *Proc. Int. Symposium on Physical Design*, Monterey, CA, April 1999, pp. 36–41.

[22] I-M. Liu, T.-L. Chou, A. Aziz, and D. F. Wong. "Zero-skew clock tree construction by simultaneous routing, wire sizing and buffer insertion," In *Proc. Int. Symposium on Physical Design*, San Diego, CA, April 2000, pp. 33–38.

[23] S. R. Nassif. "Modeling and analysis of manufacturing variations," In *Proc. Custom Integrated Circuits Conference*, San Diego, CA, May 2001, pp. 223–228.

[24] A. Takahashi, K. Inoue, and Y. Kajitani. "Clock-tree routing realizing a clock-schedule for semi-synchronous circuits," In *Proc. Int. Conf. on Computer Aided Design*, San Jose, CA, Nov. 1997, pp. 260–265.

[25] W. C. Elmore. "The transient response of damped linear networks with particular regard to wideband amplifiers," *Journal of Applied Physics*, vol. 19(no.1), pp. 155–163, Jan. 1948.

[26] Rishi Chaturvedi and Jiang Hu. "A simple yet effective merging scheme for prescribed-skew clock routing," In *Proc. Int. Conf. on Computer Design*, San Jose, CA, Oct. 2003, pp. 282–287.

[27] C.-K. Cheng, J. Lillis, S. Lin, and N. Chang. *Interconnect analysis and synthesis*. New York: Wiley Interscience, 2000.

[28] J. Chung and C.-K. Cheng. "Skew sensitivity minimization of buffered clock tree," In *Proc. Int. Conf. on Computer Aided Design*, San Jose, CA, Nov. 1994, pp. 280–283.

[29] Rishi Chaturvedi and Jiang Hu. "Buffered clock tree for high quality IC design," In *Proc. IEEE Int. Symposium on Quality Electronic Design*, San Jose, CA, March 2004, pp. 381–386.

[30] L. P. P P. van Ginneken. "Buffer placement in distributed RC-tree networks for minimal Elmore delay," In *Proc. Int. Symposium on Circuits and Systems*, New Orleans, LA, May 1990, pp. 865–868.

[31] Rishi Chaturvedi and Jiang Hu. "An efficient merging scheme for clock routing with general skew targets," In *Proc. TAU Workshop*, Austin, TX, Feb. 2004, pp. 112–118.

APPENDIX A

SAMPLE TESTCASE: IP_SAMPLE

Following is a small sample testcase from IBM, which is obtained from an actual prescribed-skew clock tree. PerUnitResistance is the resistance per unit length in ohms and PerUnitCapacitance is the capacitance per unit length in farads. Delay-target is the target delay for the sink in fermiseconds.

NumPins : 15

PerUnitResistance : 0.006000

PerUnitCapacitance : 56.000000e-17

Sink : 0

Coordinate : 2460 1895

Capacitive Load : 16.600000e-14

delay-target : 043000

Sink : 1

Coordinate : 1900 1032

Capacitive Load : 16.600000e-14

delay-target : 038000

Sink : 2

Coordinate : 2186 503

Capacitive Load : 16.600000e-14

delay-target : 034000

Sink : 3

Coordinate : 1402 647

Capacitive Load : 16.600000e-14

delay-target : 038000

Sink : 4

Coordinate : 828 2807

Capacitive Load : 16.600000e-14

delay-target : 013000

Sink : 5

Coordinate : 1132 3287

Capacitive Load : 16.600000e-14

delay-target : 010000

Sink : 6

Coordinate : 3214 1919

Capacitive Load : 16.600000e-14

delay-target : 031000

Sink : 7

Coordinate : 2428 2471

Capacitive Load : 16.600000e-14

delay-target : 026000

Sink : 8

Coordinate : 2652 1103

Capacitive Load : 16.600000e-14

delay-target : 014000

Sink : 9

Coordinate : 1454 3456

Capacitive Load : 16.600000e-14

delay-target : 023000

Sink : 10

Coordinate : 3382 2376

Capacitive Load : 16.600000e-14

delay-target : 043000

Sink : 11

Coordinate : 2393 3119

Capacitive Load : 16.600000e-14

delay-target : 028000

Sink : 12

Coordinate : 3104 912

Capacitive Load : 16.600000e-14

delay-target : 029000

Sink : 13

Coordinate : 807 1704

Capacitive Load : 16.600000e-14

delay-target : 006000

Sink : 14

Coordinate : 1091 2160

Capacitive Load : 16.600000e-14

delay-target : 000000

VITA

Name	Rishi Chaturvedi
Address	4, Brahmmanand Colony, Durgakund
	Varanasi, Uttar Pradesh, India-221005
Education	Master of Science (May 2004)
	Major - Computer Engineering
	Texas A&M University, College Station, TX.
	Bachelor of Technology (May 2000)
	Major - Electrical and Electronics Engineering
	Indian Institute of Technology (IIT)
	Kanpur, Uttar Pradesh, India - 208016
Work Experience	Asic Design Engineer (June 2000- Aug 2002)
	Transwitch Corporation, Delhi, India.

The typist for this thesis was Rishi Chaturvedi.