Buffered Clock Tree for High Quality IC Design

Rishi Chaturvedi and Jiang Hu
Department of Electrical Engineering
Texas A&M University, College Station, TX 77843
{rishi, jianghu}@ee.tamu.edu

Abstract

In ultra-deep submicron VLSI designs, clock network layout plays an increasingly important role on determining circuit quality indicated by timing, power consumption, cost, power supply noise and tolerance to process variations. In this paper, a clock tree routing algorithm is proposed to achieve any prescribed non-zero skews which are useful in reducing clock cycle time [1], suppressing power supply noise [2] and improving tolerance to process variations [3]. 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 imply cost, power consumption and vulnerability to process variations. During the clock routing, buffers are inserted simultaneously to facilitate a proper skew rate level and reduce wire snaking. The proposed algorithm is simple and fast for practical applications. Experimental results on benchmark circuits show that the proposed algorithm can reduce the total wire and buffer capacitance by 60% *over an extension of existing zero skew routing method.*

1 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 [4, 5] 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 driver to each sink which is either a flip-flop or a latch. Since clock skew is often more important than delay itself, this specifying process is often called skew scheduling [1]. It was observed long time ago that certain prescribed non-zero skew could be utilized to improve clock frequency [1, 6]. In this scenario, a skew refers to the delay difference between a certain clock sink pair. In addition to timing improvement, prescribed skews help to reduce simultaneous signal switching and power supply noise [2]. Moreover, tolerance to process variations can be improved by setting each skew value close to the center of its permissible range [3]. Therefore, prescribed non-zero skew is a very promising approach to improve circuit timing, power supply noise and reliability. Consequently, a clock routing method for prescribed non-zero skew is strongly needed. A prescribed skew routing algorithm should minimize wirelength and buffer area as well, since 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 driver is 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 driver 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.

0-7695-2093-6/04 $20.00 © 2004 IEEE

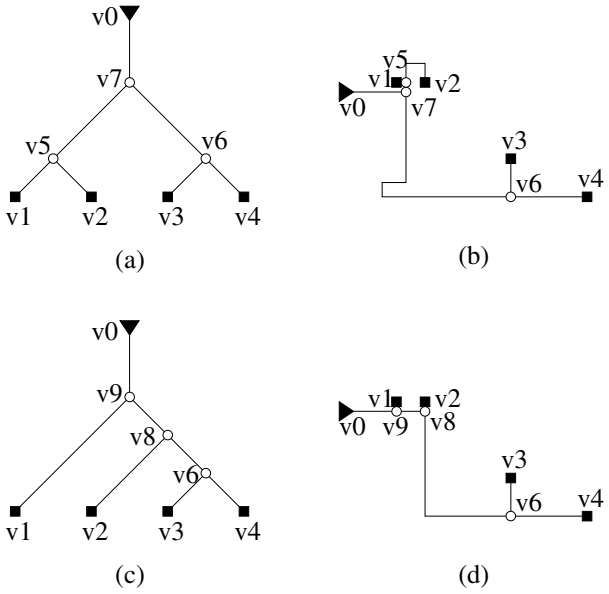

Figure 1. When delay-targets for four sinks $t_4 > t_3 >> t_2 > t_1$, traditional merging scheme may result in abstract tree in (a) and embedding in (b) with wire snakings. A different abstract tree in (c) and its layout embedding in (d) may yield less wirelength.

Most of previous works on clock network design attempt to obtain zero skew, because the skew is a lower bound for clock period time [6]. In this scenario, a more precise definition of skew is the *maximum* delay difference among all clock sinks. Early zero skew routing works include H-tree [6], top-down recursive partitioning [7] and bottom-up recursive matching method [8]. However, these methods emphasize on load balancing without evaluating actual delay. In [9], 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 [10] 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 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 [7, 8, 10] sought for a balanced abstract tree to facilitate zero skew. However, Edahiro noted in [12] 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 [12], 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 compared to choosing all subtrees in other works. The work of [12] reported one of the best wirelength results 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 [9, 10] 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 [9] 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 [13, 14] integrate skew scheduling with clock routing to exploit the useful skews. Starting with a zero skew routing tree, the work of [14] 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 [13], 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 [15–18] place buffers of the same size at nodes of the same level in the clock tree 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. However, this strategy is not applicable for non-zero skew routing which may generate unbalanced trees. Further, the increasingly significant intra-die process variations [19] request for a more general variation tolerance technique such as non-zero skew scheduling [3]. A buffered clock tree algorithm for prescribed skews is proposed in [20]. 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. We analyzed 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) 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 [12] 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.

2 Preliminary

Same as other clock routing works, we adopt the Elmore delay model for delay computation. 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.*

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 $t_1 = t_2 = ... = t_n$.

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 2(a), let the wirelength from v_i to v_j and v_k be $l_{i,j}$ and $l_{i,k}$, respectively. The delay from v_i to v_j and v_k are:

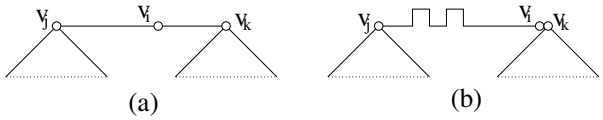

Figure 2. Examples of merging subtrees without wire snaking in (a) and with wire snaking when delay-target t_j at v_j is significantly greater than delay-target t_k at v_k in (b).

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

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:

$$d_{i,j} - d_{i,k} = t_j - t_k \quad (2)$$

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

$$t_i = t_j - d_{i,j} = t_k - d_{i,k} \quad (3)$$

Since the delay-targets are propagated bottom-up based on the above equation, the skew specifications can be enforced by only considering Equation (2) 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) is met. The actual wirelength of $l_{i,j}$ can be obtained by solving the following equation.

$$\frac{1}{2} r c l_{i,j}^2 + r l_{i,j} C_j = t_j - t_k \quad (4)$$

The method of using wirelength greater than $l_{j,k}$ is called wire snaking [9] which is demonstrated in Figure 2(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 $t_b + R_b C_i$ 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.

3 Algorithm

The top-level framework of our algorithm is similar to Edahiro's NS algorithm [12], however, we propose a merging scheme and a buffering method that are designed particularly for prescribed non-zero skew clock routing. This merging scheme is also integrated with DME embedding as in [12].

3.1 The Merging Scheme

Most of previous merging schemes [8, 12] 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 (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 (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.

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

Figure 3. Algorithm of the merging selection scheme.

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 [12] 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). 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(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 3. 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.

3.2 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. 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 [21], 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 [18, 22] or a greedy approach [20]. 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 [20].

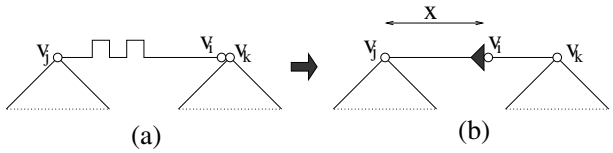

Figure 4. Buffer insertion to reduce wire snaking. Delay target $t_j > t_k$.

Sometimes a buffer may be inserted to reduce wire snaking. 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 root v_j. Please note this conclusion is contrary to the case in signal routing where buffers are usually employed to reduce delay [6]. For the example in Figure 4, since $t_j > t_k$, a buffer is inserted to drive the branch with T_j as in Figure 4(b).

Next, the buffer location needs to be decided in addition to the merging node v_i location. The work of [20] 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 4(b). In Figure 4(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.

3.3 Complexity

When integrated with the DME embedding as in [12], 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 3. 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 $O(n^2)$.

4 Experiment

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 [11] 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. For comparison, the NS(Nearest-neighbor Selection) algorithm proposed in [12] is extended for non-zero skew targets and combined with the same buffering scheme as ours.

The experimental results are shown in Table 1. Since both algorithms deliver the same prescribed non-zero

Table 1. Comparison of our buffered clock tree routing and an extension to the NS algorithm [12].

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%

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 1. Note that our buffered clock routing is not only effective but also fast for practical applications.

5 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 buffered clock routing algorithm based on the maximum delay-target ordering merging. 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. More accurate delay models will be employed for prescribed skew routing in future work.

References

[1] J. P. Fishburn. Clock skew optimization. *IEEE Transactions on Computers*, C-39:945–951, July 1990.

[2] W.-C. D. Lam, C.-K. Koh, and C.-W. A. Tsao. Power supply noise suppression via clock skew scheduling. In *Proc. ISQED*, pages 355–360, 2002.

[3] I. S. Kourtev and E. G. Friedman. Clock skew scheduling for improved reliability via quadratic programming. In *Proc. ICCAD*, pages 239–243, 1999.

[4] 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. DAC*, pages 168–171, 2000.

[5] S. Zanella, A. Nardi, A. Neviani, M. Quarantelli, S. Saxena, and C. Guardiani. Analysis of the impact of process variations on clock skew. *IEEE Transactions on Semiconductor Manufacturing*, 13(4):401–407, November 2000.

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

[7] M. A. B. Jackson, A. Srinivasan, and E. S. Kuh. Clock routing for high-performance ICs. In *Proc. DAC*, pages 573–579, 1990.

[8] A. B. Kahng, J. Cong, and G. Robins. High-performance clock routing based on recursive geometric matching. In *Proc. DAC*, pages 322–327, 1991.

[9] R.-S. Tsay. Exact zero skew. In *Proc. ICCAD*, pages 336–339, 1991.

[10] T.-H. Chao, Y.-C. Hsu, J.-M. Ho, K. D. Boese, and A. B. Kahng. Zero skew clock routing with minimum wirelength. *IEEE Transactions on Circuits and Systems - Analog and Digital Signal Processing*, 39(11):799–814, November 1992.

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

[12] M. Edahiro. A clustering-based optimization algorithm in zero-skew routings. In *Proc. DAC*, pages 612–616, 1993.

[13] C.-W. A. Tsao and C.-K. Koh. UST/DME: a clock tree router for general skew constraints. In *Proc. ICCAD*, pages 400–405, 2000.

[14] J. G. Xi and W. W.-M. Dai. Useful-skew clock routing with gate sizing for low power design. *Journal of VLSI Signal Processing*, 16(2/3):163–179, Jun./Jul. 1997.

[15] S. Pullela, N. Menezes, J. Omar, and L. T. Pillage. Skew and delay optimization for reliable buffered clock trees. In *Proc. ICCAD*, pages 556–562, 1993.

[16] J. G. Xi and W. W.-M. Dai. Buffer insertion and sizing under process variations for low power clock distribution. In *Proc. DAC*, pages 491–496, 1995.

[17] A. Vittal and M. Marek-Sadowska. Power optimal buffered clock tree design. In *Proc. DAC*, pages 230–236, 1996.

[18] 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. ISPD*, pages 33–38, 2000.

[19] S. R. Nassif. Modeling and analysis of manufacturing variations. In *Proc. CICC*, pages 223–228, 2001.

[20] A. Takahashi, K. Inoue, and Y. Kajitani. Clock-tree routing realizing a clock-schedule for semi-synchronous circuits. In *Proc. ICCAD*, pages 260–265, 1997.

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

[22] J. Chung and C.-K. Cheng. Skew sensitivity minimization of buffered clock tree. In *Proc. ICCAD*, pages 280–283, 1994.