[15] S. S. Leung and M. A. Shanblatt, *ASIC System Design With VHDL: A Paradigm*. Norwell, MA: Kluwer, 1990.
[16] E. Yeo, S. Augsburger, W. Rhett Davis, and B. Nikolie, "500 Mb/s soft-output Viterbi decoder," in *Proc. 28th Eur. Solid-State Circuits Conf.*, Florence, Italy, Sep. 24–26, 2002, pp. 523–526.
[17] V. S. Gierenz, O. Weiss, T. G. Noll, I. Carew, J. Ashley, and R. Karabed, "A 550 Mb/s radix-4 bit-level pipelined 16-state 0.25-$\mu$m CMOS Viterbi decoder," in *Proc. IEEE Int. Conf. Application-Specific Systems, Architectures, and Processors (ASAP'00)*, Boston, MA, Jul. 10–12, 2000, pp. 195–201.
[18] D. E. Hocevar and A. Gatherer, "Achieving flexibility in a Viterbi decoder DSP coprocessor," in *Proc. Vehicular Technology Conference*, vol. 5, Boston, MA, Sep. 24–28, 2000, pp. 2257–2264.
[19] G. Kang and P. Zhang, "The implementation of Viterbi decoder on TMS320C6201 DSP in W-CDMS systems," in *Proc. Int. Conf. Communication Technology*, vol. 2, Beijing, China, Aug. 21–25, 2000, pp. 1693–1696.

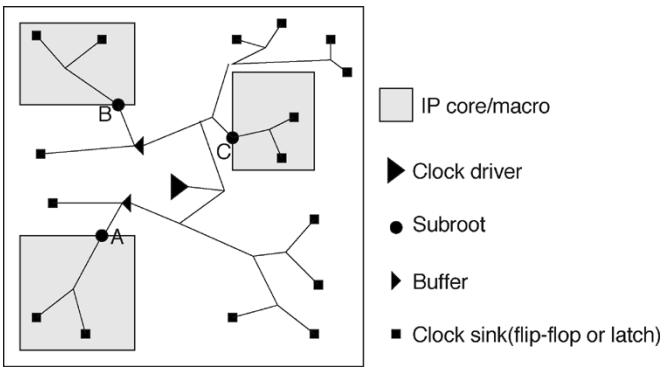


Fig. 1. Clock network in a hierarchical/SoC design.

# An Efficient Merging Scheme for Prescribed Skew Clock Routing

Rishi Chaturvedi and Jiang Hu

***Abstract*—In ultra-deep submicron very large-scale integration (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 brief, a new merging scheme is proposed for prescribed nonzero skew routings which are useful in reducing clock cycle time, suppressing power-supply noise, and improving tolerance to process variations. This technique is simple and easy to implement for practical applications. Experimental results on benchmark circuits with both buffered and unbuffered routings exhibit large improvement on wirelength and buffer cost compared with other existing works.**

***Index Terms*—Clocks, design automation, very large-scale integration.**

## I. Introduction

In synchronous very large-scale integration (VLSI) circuits, since the pace of almost every data transfer is coordinated by clock signals, the quality of clock networks has a vital influence to the circuit timing performance. Moreover, as the clock network is perhaps the largest interconnect net, a clock network with excessive wirelength may lead to severe problems on power consumption, power/ground supply noise, process variations, and thermal issues. Thus, a clock network with elaborated timing characteristics and minimal wirelength is crucial for ultra deep submicron VLSI designs.

It was observed a long time ago that certain prescribed nonzero skew could be utilized to improve clock frequency [1]. 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]. Consequently, a clock routing method for prescribed nonzero skew is strongly needed. In hierarchical/system-on-chip (SoC) designs, even if only zero skew is sought, the prescribed skew routing technique is useful. In this scenario, the entire clock network can be divided into two levels as shown in Fig. 1: 1) local clock networks within each block (IP core/macro) and 2) the chip level global clock network that connects the subroots of the local networks and sinks outside of any blocks. In practice, the global network and each local network are usually designed separately and the subroot-sink delay in one local network is quite likely to be different from that in another local network. Thus, the global clock network needs to be constructed according to nonzero skew specifications.

Clock tree routing is usually a process that recursively merges a set of subtrees, which are clock sinks initially, in a bottom-up fashion. 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. There are two major decision makings in this clock tree routing process: 1) *merging scheme* that tells which subtrees should be merged together and 2) *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.

Most previous works on clock-network design attempt to obtain zero skew, because the skew is a lower bound for clock period time. In [4], 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 deferred merge embedding (DME) algorithm was developed in [5]. Instead of committing a merging node to particular location immediately, DME maintains *merging segment* for each merging node and defers the decision on the exact merging location later when it is clear how to minimize wirelength. The DME embedding is extended to handle bounded skew clock routing in [6]. Since the skew in [6] can be any value within a global upperbound, it is quite different from the local-pair-wise prescribed skews discussed in [2], [3]. In [7], a nearest neighbor based merging selection is integrated with DME embedding.

In contrast to numerous works on zero skew clock routing, there are very few works reported on prescribed nonzero skew routing despite its great importance. Perhaps this is due to the misconception that existing zero skew routing techniques can be applied to nonzero skew directly. Indeed, the layout embedding techniques originally designed for zero skew [4], [5] can be adopted directly to achieve nonzero skews. However, zero skew driven *merging schemes* do not necessarily work well for nonzero skew clock routing. In fact, we discover that huge wirelength is generated through traditional merging schemes. This is especially true when the differences among delay-targets are large so that a lot of wire snakings [4] are incurred. The example in Fig. 2 illustrates

Manuscript received August 12, 2004; revised October 14, 2004. This work is supported in part by Semiconductor Research Corporation under Contract 2004-TJ-1205.

R. Chaturvedi is with the Analog Devices Inc., Wilmington, MA 01887 USA.
J. Hu is with the Department of Electrical Engineering, Texas A&M University, TX 77843 USA (e-mail: jianghu@ee.tamu.edu).

Digital Object Identifier 10.1109/TVLSI.2005.848821

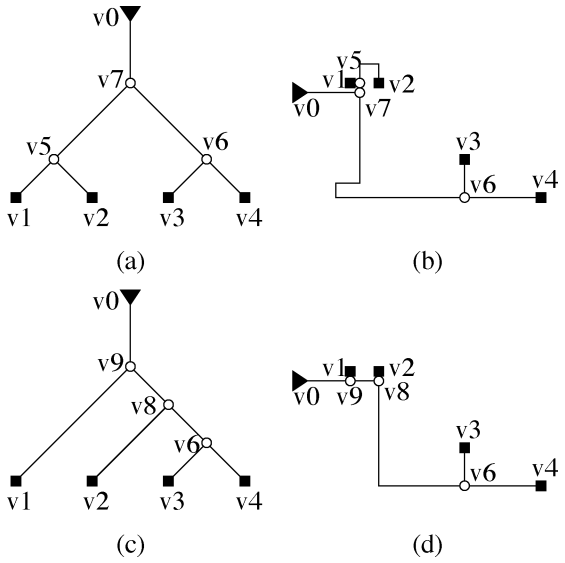


Fig. 2. When delay targets for four sinks $\boldsymbol{t}_4 > \boldsymbol{t}_3 \gg \boldsymbol{t}_2 > \boldsymbol{t}_1$, traditional merging scheme may result in an abstract tree in (a) and embedding in (b) with wire snakings. A different abstract tree in (c) and its layout embedding in (d) may yield less wirelength.

that different merging schemes may provide different wirelength for nonzero skew clock routing.

In [8], an incremental scheduling algorithm is proposed and combined with the DME embedding for a certain abstract tree. In an early work [9] of prescribed skew routing, the nearest neighbor based merging scheme of [7] is extended to a merging based on minimum merging cost which includes wire detours due to delay target imbalance. Recently, a clock network synthesis method [10] is developed such that clock signal delay to each clock sink is bounded by distinctive lower and upper limit.

The goal of this work is to develop a merging scheme for prescribed skew routing with minimum wirelength. We analyzed the interactions among skew targets, sink location proximities and capacitive load balance in clock routing. According to this analysis, we propose a maximum delay-target-based merging scheme which is integrated with the DME embedding to achieve general prescribed skews. The total wirelength is minimized to restrict cost and power consumption. The proposed technique is simple and fast for practical applications. Even though the work of [8] is also aimed to achieve general skew targets, it emphasizes more on incremental scheduling while our contribution is mostly on the merging scheme. The two works are suitable for different methodology flows. We compared our method with other existing methods [7], [9] on benchmark circuits. The experimental results show that our method can lead to significant wirelength reduction.

## II. Preliminary

Same as other clock routing works, we adopt the Elmore delay model for delay computation in this brief. A merging scheme only decides a merging order and can be applied with any delay models. The problem we will solve is formally stated as follows.

**Minimum Cost Prescribed Skew Clock Routing Problem:** Given a set of clock sinks $V = \{v_1, v_2, \ldots v_n\}$, load capacitance $C_i$ for each sink $v_i \in V$, skew specifications $q_{i,j}$ for every pair of sinks $v_i, v_j \in V$, find a rooted Steiner tree with clock sinks as leaf nodes such that the total wirelength is minimized and 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 $v_i, v_j \in V$.

The minimum required number of skew specifications for $n$ nodes is $n-1$. The others can be derived from the $n-1$ specifications. The skew

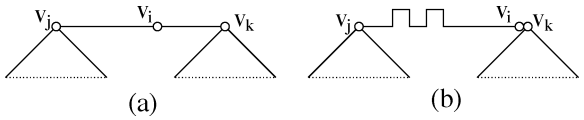


Fig. 3. Examples of merging subtrees without wire snaking in (a) and with wire snaking when delay target $\boldsymbol{t}_j$ at $\boldsymbol{v}_j$ is significantly greater than delay target $\boldsymbol{t}_k$ at $\boldsymbol{v}_k$ in (b).

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. 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 $v_i \in V$.

Now we generalize the concept of a 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, 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 Fig. 3(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

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

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 (2) without checking delays at sink nodes.

The minimum feasible wirelength for the merging is the Manhattan distance $l_{j,k}$ between $v_j$ and $v_k$. 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 $l_{i,j} > l_{j,k}$. The actual wirelength of $l_{i,j}$ can be obtained by solving the following:

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

The method of using a wirelength greater than $l_{j,k}$ is called wire snaking [4] which is demonstrated in Fig. 3(b).

Since the top-level framework of our prescribed skew clock routing method is very similar to Edahiro's zero skew routing algorithm in [7], we briefly review the basic version of Edahiro's algorithm which is called nearest-neighbor selection (NS) algorithm. The first phase is a bottom-up recursive subtree merging process, which constructs the abstract tree and finds the merging segments [5] for each merging node. In each step, only one pair of subtrees (initially clock sinks), which have the minimum distance between their roots, is selected to be merged. The nearest neighbor is found using Delaunay triangulation. After a merging, the two subtrees are replaced by the newly created subtree. This process is repeated recursively till there is only one subtree left. The second phase is a top-down tree traversal that identifies one location on each merging segment such that the total wirelength is minimized. This phase is the same as the top-down phase in DME algorithm [5]. In the same paper [7], Edahiro also presented a clustering based speed-up version of the NS algorithm which is notated as CL (clustering based).

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

Fig. 4. Algorithm of the merging selection scheme.

## III. Algorithm

### A. Merging Scheme

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 Fig. 2. Assume that the given delay targets are quite different from each other and they follow the inequality $t_1 < t_2 \ll t_3 < t_4$, 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 Fig. 2(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 (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. According to (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.

We further illustrate the advantage of this maximum delay-target ordered merging through the example in Fig. 2. In Fig. 2(d), we first merge $T_3$ with $T_4$ to obtain subtree $T_6$ rooted at $v_6$, as $T_4$ 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 Fig. 2(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 Fig. 2(d) and the merging $T_1 + T_2 \rightsquigarrow v_5$ in Fig. 2(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 [7] 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. A subtree needs to be merged to another subtree that is not only nearby but also with similar delay target. We adopt 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 (4) to include the extra wirelength due to wire snaking. Hence, we choose the companion subtree, which will lead to the minimum *merging cost*. In [9], the merging cost is employed as the primary criterion for merging ordering. In contrast, it plays an auxiliary role in our proposed merging scheme.

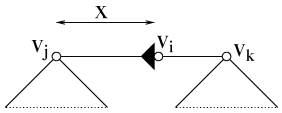


Fig. 5. Buffer insertion along with layout embedding.

The algorithm description for this merging scheme is given in Fig. 4. 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. 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.

### B. Buffered Clock Tree

In clock routing, buffers are usually inserted to ensure desired signal slew rate. Therefore, we validate our merging scheme in a context of buffered clock tree. Similar as other existing works [9], [11], *RC* switch model is employed for buffers. In a buffered clock tree, a proper slew rate can be achieved by enforcing a maximum load capacitance constraint $C_{\max}$ which a buffer/driver can drive. Since the output slew rate of a buffer/driver is mainly determined by its load capacitance, 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 [11] or a greedy approach [9]. 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 [9] which inserts buffers whenever downstream load capacitance may exceed $C_{\max}$. The buffer cost is considered together with wire cost in term of its capacitance.

The buffer location needs to be decided in addition to the merging node $v_i$ location. The work of [9] 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 Fig. 5. In Fig. 5, 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.

TABLE I
SKEW TARGET (ps) DISTRIBUTIONS

| Case | 0-20 | 20-40 | 40-60 | 60-80 | 80-100 |
|---|---|---|---|---|---|
| r1 | 46 | 54 | 50 | 48 | 69 |
| r2 | 111 | 91 | 123 | 98 | 175 |
| r3 | 176 | 148 | 106 | 152 | 280 |
| r4 | 305 | 250 | 334 | 387 | 647 |
| r5 | 522 | 525 | 521 | 593 | 940 |

### C. Complexity

When integrated with the DME embedding as in [7], 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 Fig. 4. Please note that line 1 of Fig. 4 takes only $O(\log n)$ time assuming that there are very few sinks with same delay target in prescribed skew design. 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 $O(n^2)$. When there are a lot of sinks with same delay target, the minimum merging cost pair needs to be identified among all the sinks with same delay target to break the tie. In the worst case as in the zero skew designs, $O(n^2)$ time is needed to identify the minimum merging cost pair.

## IV. EXPERIMENT

We implemented the proposed clock tree routing algorithm in C and experiments are performed on a PC with 1.7-GHz Pentium 4 microprocessor and 512 Mb memory. The benchmark circuits are r1–r5 downloaded from the GSRC Bookshelf (http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/BST/). The delay targets are generated through running the BST [6] code with a global skew bound of 100 ps and taking the nonzero skew results. The BST code is also downloaded from the GSRC Bookshelf. The skew target distributions from BST are listed in Table I. Each column indicates a skew range with respect to the minimum delay target among all sinks. Each entry shows the number of sinks in a specific skew range.

We compared the following four clock routing algorithms with different merging schemes.

- NS+: The nearest-neighbor selection algorithm in [7] using the minimum distance merging. The DME is implemented with trivial extension such that (2) instead of zero skew constraint is enforced in finding merging segment locations.
- MIC [9]: 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.
- MAT: Choose the maximum delay-target subtree first and find its companion subtree that is its nearest neighbor (with minimum distance).
- 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.

The experimental results for nonzero skew targets of unbuffered clock trees are shown in Table II. Since these clock routing algorithms all deliver the same prescribed nonzero skews, we only report the total wirelength here. The relative wirelength with respect to MAT-MIC are also listed. 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+. Simply extending zero skew routing as NS+ usually doubles the wirelength compared to our proposed MAT-MIC scheme.

TABLE II
EXPERIMENTAL RESULTS OF NON-ZERO PRESCRIBED SKEW UNBUFFERED CLOCK ROUTINGS

| Case | #sinks | Merging | Wirelength | (relative) | CPU(sec) |
|---|---|---|---|---|---|
| r1 | 267 | NS+ | 2477975 | (2.14) | 1 |
| | | MIC | 1515972 | (1.31) | 4 |
| | | MAT | 1491833 | (1.29) | 1 |
| | | MAT-MIC | 1160145 | (1.00) | 1 |
| r2 | 598 | NS+ | 4935718 | (2.20) | 20 |
| | | MIC | 2936488 | (1.31) | 44 |
| | | MAT | 2656833 | (1.18) | 1 |
| | | MAT-MIC | 2243723 | (1.00) | 1 |
| r3 | 862 | NS+ | 6847146 | (2.40) | 56 |
| | | MIC | 3927066 | (1.38) | 118 |
| | | MAT | 3242982 | (1.14) | 1 |
| | | MAT-MIC | 2851177 | (1.00) | 1 |
| r4 | 1903 | NS+ | 15123351 | (2.60) | 462 |
| | | MIC | 7358685 | (1.26) | 1250 |
| | | MAT | 7283887 | (1.25) | 3 |
| | | MAT-MIC | 5819055 | (1.00) | 4 |
| r5 | 3101 | NS+ | 23013115 | (2.61) | 2572 |
| | | MIC | 11076659 | (1.26) | 6234 |
| | | MAT | 10695010 | (1.21) | 8 |
| | | MAT-MIC | 8810532 | (1.00) | 11 |

TABLE III
EXPERIMENTAL RESULTS OF ZERO SKEW UNBUFFERED CLOCK ROUTINGS

| | NS [7] | | MAT | |
|---|---|---|---|---|
| Case | Wirelen | CPU(s) | Wirelen | CPU(s) |
| r1 | 1573223 | 3 | 1289459 | 2 |
| r2 | 2774181 | 28 | 2587492 | 21 |
| r3 | 3683852 | 92 | 3282424 | 64 |
| r4 | 7224428 | 879 | 6643357 | 664 |
| r5 | 10784090 | 3755 | 9839246 | 2853 |
| Ave. improvement | | | 9.2% | 24.2% |

The previous work MIC [9] still yields 26%–31% more wirelength than MAT-MIC. Using MAT alone is not adequate and MAT-MIC gives the best result. The CPU time for each routing algorithm are shown in the rightmost column. Note that our merging scheme is not only effective but also fast for practical applications. In particular, our method is much faster than that of MIC [9].

Even though our merging scheme is designed primarily for nonzero skews, we also compared it for zero skew routings with the NS algorithm in [7]. The results are listed in Table III. Since the delay-target differences among subtrees in zero skew routing are normally not large, the MIC technique is rarely useful. Therefore, we only report the MAT results here. The results in Table III show that our merging scheme can make improvement on wirelength even for zero skew routings even though the improvement is not as large as in nonzero skew routings.

In zero skew routings and NS+ and MIC for nonzero skew routings, $O(n^2)$ time is needed to find the minimum merging cost pair among all pairs. In contrast, in MAT and MAT-MIC for nonzero skew routings, $O(\log n)$ time is required for finding the maximum delay target subtree and $O(n)$ is spent on finding the companion subtree. This explains the runtime differences in Table II and Table III. Therefore, MAT can improve runtime for nonzero skew routing in addition to reducing wirelength.

Our merging scheme is also validated in buffered clock tree routings with the experimental results shown in Table IV. We 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 large improvement. Since the maximum load capacitance is restricted in the buffered clock tree, the driver/buffer to buffer/sink delay is typically less than 175 ps. According to the first

TABLE IV
EXPERIMENTAL RESULTS OF NON-ZERO PRESCRIBED SKEW BUFFERED CLOCK ROUTINGS. TOTAL CAPACITANCE INCLUDES WIRE AND BUFFER CAPACITANCE IN pF. THE CPU TIME IS IN SECONDS

| Case | Algorithm | Wirelen | #bufs | Total cap | CPU |
|---|---|---|---|---|---|
| r1 | NS+ | 2917614 | 49 | 59.5 | 1 |
| | MAT-MIC | 1293616 | 13 | 26.2 | 1 |
| r2 | NS+ | 5881313 | 93 | 119.8 | 13 |
| | MAT-MIC | 2541324 | 25 | 51.4 | 1 |
| r3 | NS+ | 7287088 | 112 | 148.4 | 42 |
| | MAT-MIC | 3265058 | 28 | 66.0 | 1 |
| r4 | NS+ | 16276930 | 474 | 331.1 | 474 |
| | MAT-MIC | 6659865 | 62 | 134.6 | 3 |
| r5 | NS+ | 24933092 | 1968 | 507.7 | 1968 |
| | MAT-MIC | 9860004 | 99 | 199.5 | 10 |
| Ave. improvement | | 59.6% | 69.0% | 60.0% | 99.3% |

order estimation metric in [1], slew rate for step input is roughly $\ln 9 \times Elmore\_delay$. Therefore, the slew rate at the sinks for step input is usually less than 385 ps. Please note that this metric is a conservative estimation. For ramp input with 100 ps slew rate, the slew rate at the sinks is usually no greater than 400 ps according to the metric developed in [12]. The CPU time are shown in the rightmost column of Table IV.

## V. CONCLUSION

Even though traditional zero skew routing methods can be applied to achieve nonzero 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 maximum delay-target-based merging scheme for general prescribed skew routings. Experimental results on benchmark circuits show that the proposed technique is very effective and efficient on minimizing clock network size for prescribed skews.

## REFERENCES


[1] H. B. Bakoglu, *Circuits, Interconnections and Packaging for VLSI*. Reading, MA: Addison-Wesley, 1990.

[2] W.-C. D. Lam, C.-K. Koh, and C.-W. A. Tsao, "Power supply noise suppression via clock skew scheduling," in *Proc. IEEE Int. Symp. Quality Electronic Design*, 2002, pp. 355–360.

[3] I. S. Kourtev and E. G. Friedman, "Clock skew scheduling for improved reliability via quadratic programming," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1999, pp. 239–243.

[4] R.-S. Tsay, "Exact zero skew," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1991, pp. 336–339.

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

[6] J. Cong, A. B. Kahng, C.-K. Koh, and C.-W. A. Tsao, "Bounded-skew clock and Steiner routing," *ACM Trans. Design Automation of Electronic Systems*, vol. 3, no. 3, pp. 341–388, Jul. 1998.

[7] M. Edahiro, "A clustering-based optimization algorithm in zero-skew routings," in *Proc. ACM/IEEE Design Automation Conf.*, 1993, pp. 612–616.

[8] C.-W. A. Tsao and C.-K. Koh, "UST/DME: a clock tree router for general skew constraints," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 2000, pp. 400–405.

[9] A. Takahashi, K. Inoue, and Y. Kajitani, "Clock-tree routing realizing a clock-schedule for semi-synchronous circuits," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1997, pp. 260–265.

[10] S. Held, B. Korte, J. Maßberg, M. Ringe, and J. Vygen, "Clock scheduling and clock tree construction for high performance ASICs," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 2003, pp. 232–239.

[11] J. Chung and C.-K. Cheng, "Skew sensitivity minimization of buffered clock tree," in *Proc. IEEE/ACM Int. Conf. Computer-Aided Design*, 1994, pp. 280–283.

[12] C. V. Kashyap, C. J. Alpert, F. Liu, and A. Devgan, "Closed-form expressions for extending step delay and slew metrics to ramp inputs for *RC* trees," *IEEE Trans. Computer-Aided Des. Integr. Circuits Syst.*, vol. 23, no. 4, pp. 509–516, Apr. 2004.


# Comparison of High-Performance VLSI Adders in the Energy-Delay Space

Vojin G. Oklobdzija, Bart R. Zeydel, Hoang Q. Dao, Sanu Mathew, and Ram Krishnamurthy

***Abstract*—In this paper, we motivate the concept of comparing very large scale integration adders based on their energy-delay characteristics and present results of our estimation technique. This stems from a need to make appropriate selection at the beginning of the design process. The estimation is quick, not requiring extensive simulation or use of computer-aided design tools, yet sufficiently accurate to provide guidance through various choices in the design process. We demonstrate the accuracy of the method by applying it to examples of high-performance 32- and 64-b adders in 100- and 130-nm CMOS technologies.**

***Index Terms*—Adders, digital arithmetic, digital circuits, energy-delay optimization.**

## I. INTRODUCTION

In the very large scale integration (VLSI) design process, selection of the initial topology expected to yield a desired performance in the allotted power budget is the most important step taken. However, the exact performance and power will be known only after a time consuming design and simulation process is completed. Therefore, the validity of the initial selection will not be known until late in the design process. Going back and forth between several choices is often prohibited by design schedule, making it impossible to correct mistakes committed at the beginning. Therefore an uncertainty always remains as to whether a higher performance or lower power could have been achieved using a different topology. This problem is aggravated by a lack of proper delay and power estimation techniques that are guiding development of computer arithmetic algorithms. The majority of algorithms used today are based on out-dated methods of counting the number of logic gates on the critical path, producing inaccurate and misleading results. The importance of transistor sizing, load effects and power are not taken into account by most.

Knowles has shown how different adder topologies may influence fan-out and wiring density, thus, influencing design decisions and yielding better area/power tradeoffs than known cases [1]. This work further emphasized the disconnect existing between algorithms and implementation. In our previous work, the importance of fan-in and fan-out effects on the critical path was demonstrated [2]. This led to similar conclusions expressed in the logical effort (LE) method of Sutherland and Sproull [3] regarding critical-path delay estimation. This method has been introduced into common practice by Harris [4]. To gain confidence in the method we compared LE delay estimate of various VLSI adders to simulation results [5] obtained using H-SPICE [17]. The matching was well under 10% in most cases (Table I). However, given that delay and energy can be traded against each other, inclusion of energy in the analysis might have resulted in different ranking.

Manuscript received July 20, 2004; revised December 6, 2004. This work was supported in part by the SRC under Research Grant 931.001 and in part by California MICRO 03-069.

V. G. Oklobdzija, B. R. Zeydel, and H. Q. Dao are with the Advanced Computer Systems Engineering Laboratory (ACSEL), Electrical and Computer Engineering Department, University of California, Davis, CA 95616 USA (e-mail: zeydel@acsel-lab.com).

S. Mathew and R. Krishnamurthy are with the Microprocessor Research Laboratories, Intel Corporation, Hillsboro, OR 97124 USA.

Digital Object Identifier 10.1109/TVLSI.2005.848819