An Exact Zero-Skew Clock Routing Algorithm

Ren-Song Tsay, *Member, IEEE*

Abstract—**In this paper we present an exact zero-skew clock routing algorithm using the Elmore delay model. The results have been verified with accurate waveform simulation. We first review a linear time delay computation method. A recursive bottom–up algorithm is then proposed for interconnecting two zero-skewed subtrees to a new tree with zero skew. The algorithm can be applied to single-staged clock trees, multistaged clock trees, and multi-chip system clock trees. The approach is ideal for hierarchical methods of constructing large systems. All subsystems can be constructed in parallel and independently, then interconnected with exact zero skew. Extensions to the routing of optimum nonzero-skew clock trees (for cycle stealing) and multiphased clock trees are also discussed.**

I. Introduction

IN THIS PAPER we propose an exact zero-skew clock routing algorithm for optimizing the timing performance of synchronous digital systems. Clock skew is defined as the maximum difference of the delays from the clock source to the clock pins on latches. Optimization of the clock skew can dramatically reduce the system's cycle time, and, hence, the timing performance. In contrast, improper clock skew may sometimes cause clock hazard and system malfunction [6]. The following equation summarizes the relationship of the clock period P, clock skew s, worst-case data path delay $d_{\max}$, and other offset constant P_o for the condition of proper timing:

$$P = s + d_{\max} + P_o.$$

Note that P_o is a constant that includes data setup time, latch active time, and other possible offset factors such as safety margins, for example. The latch active time is the lag time for the data to be latched in after the latch is triggered by a clock signal.

It is clear from the equation that to reduce the cycle time P, it is necessary to minimize the skew s, besides the minimization of the worst-case data delay $d_{\max}$, on the combinational logics. As interconnection delay is becoming more dominating and design size is becoming larger, the clock skew is also becoming more significant in terms of performance optimization.

Many heuristics for clock routing have been proposed in the past. H-tree structures [1], [4], [10], [7] are the most widely used, especially in systolic array designs. A generalization of an H-tree that hierarchically connects the median points is proposed in [8]. A further improvement is done by bottom–up pairwise connections which construct a perfect length balanced tree [9]. However, all these heuristics focus only on wire length balancing, rather than the real objective of balancing clock delay. These approaches are not effective enough for tight skew optimization, as encountered in many high-performance designs nowadays. In contrast, what we propose is an exact algorithm that balances the clock delays directly. It is a general approach that takes into account uneven loading and buffering effects.

The outline of this paper is as follows. We first study how to compute signal delays efficiently on an *RC* tree. An *RC* tree is a connected acyclic undirected graph, with each branch associated with a resistance value and each node associated with a capacitance value.

Next, we discuss how a clock tree is modeled as an *RC* tree for delay analyses. In general, clock trees are classified into two types. The first type is *single-staged* clock trees in which clock pins are driven directly from a clock source. In order to reduce phase delays (the maximum delay from the clock source to a clock pin) and supply sufficient driving currents, usually several levels of buffers are added to create a multistaged clock tree. Thus the second type is called *multistaged* clock trees, in which the clock pins are driven from intermediate buffers, and the buffers are driven by either other buffers or the clock source. A multichip system clock tree is basically a multistaged clock tree, except that the clock pins are scattered on many chips (or cards).

The zero-skew algorithm is then presented. Based on a lumped delay model and the delay computation method, we found that any two zero-skewed subtrees can be merged into a tree with zero skew by tapping the connection to a specific location of each subtree. Basically, it is a recursive bottom–up algorithm.

Finally, we present experimental results of the zero-skew algorithm and comparisons with the wire length balancing heuristics [9]. We also discuss extensions to clock routing problems that require specific clock skew values (for cycle stealing), and to the problems of multiphase clock and optical skew.

II. Linear Time Hierarchical Delay Computation

We adopt the commonly used Elmore delay model [5], [13], [2] to calculate the signal traveling time from a clock source to each clock pin. We modify the method proposed in [13] and have a hierarchical method for computing de-

Manuscript received May 2, 1991; revised February 7, 1992. This paper was recommended by Associate Editor M. Marek-Sadowska.

R.-S. Tsay was with the IBM T. J. Watson Research Center, Yorktown Heights, NY 10598. He is now with Arcsys, Inc., Santa Clara, CA 95054.

IEEE Log Number 9201136.

0278-0070/93$03.00 © 1993 IEEE

lays in a bottom-up fashion, which is the key to our zero-skew algorithm.

Rubinstein *et al.* [13] proposed a delay computation method using common path resistances of all node pairs and node capacitances. The time complexity is at least quadratic, due to the common path resistance calculation of all node pairs. Instead, we use branch resistance and total subtree capacitance for delay calculation. The new method is of linear time complexity.

To develop the algorithm, we first define a few terms. Let T represent an RC tree with every node associated with an index. We always assume the index of the root is 0. A *predecessor* of node i is a node residing on the unique path between the root and node i, but excluding node i itself. An *immediate predecessor* of node i is a predecessor of node i with no other nodes between them. Similarly, a *successor* of node i is the set of nodes which have node i as one of their predecessors. An *immediate successor* of node i is a successor of node i with no other nodes in between. The root is the node with no predecessor, and the leaf nodes are the nodes with no successors. A subtree T_i is defined as the subtree of T formed by the node i and its successors. Since T is a tree, there is only one unique edge between a node and its predecessor. So we simply define branch i as the edge between node i and its immediate predecessor.

Let c_i be the node capacitance of node i and r_i be the resistance of branch i. For convenience, if node i is the root, we set $r_i = 0$. Define $IS(i)$ as the set of all immediate successors of node i. Then the total subtree capacitance C_i of T_i is defined recursively as

$$C_i = c_i + \sum_{k \in IS(i)} C_k.$$

The above equation suggests that the subtree capacitance can be computed in a depth-first search manner. The capacitance of the subtree rooted from a node can be computed from its own node capacitance and the summation of the subtree capacitance of its immediate successors. Hence a recursive bottom-up algorithm can be used to compute the subtree capacitance of each node. The procedure of a recursive depth-first capacitance (DFC) calculation routine is outlined below. Note that a single call to $DFC(0)$ will compute the subtree capacitance of each node in a bottom-up fashion.

procedure DFC(i)
1. $C_i = c_i$;
2. **for each** $j \in IS(i)$ **do**
3. DFC(j);
4. $C_i = C_i + C_j$;
5. **end for**

end procedure

The time complexity of the depth-first search procedure is linear in terms of the number of edges [3]. Since for a tree the number of edges is $|N| - 1$, hence the time complexity of computing the subtree capacitance of every node in $\theta(|N|)$, where $|N|$ is the number of nodes on T.

To calculate the delay, we first define N as the collection of all nodes on the tree T and $N(i, j)$ as the collection of nodes on the path between node i and node j, excluding i but including node j. The delay to a leaf node i can be calculated by the following formula:

$$t_{0i} = \sum_{n \in N(0,i)} r_n C_n$$

As a generalization, we can compute the "delay time" between any two nodes i and j; by the following formula, assuming i is a predecessor of j.

$$t_{ij} = \sum_{n \in N(i,j)} r_n C_n.$$

It can be shown easily that if i is an intermediate node between node k and node j, then

$$t_{kj} = t_{ki} + t_{ij}. \tag{1}$$

Suppose that node k is the root (i.e., $k = 0$); then we have

$$t_{0j} = t_{0i} + t_{ij}$$

since there is only one edge between node i and j and $t_{ij} = r_j C_j$. Hence,

$$t_{0j} = t_{0i} + r_j C_j.$$

This equation suggests that we can easily calculate the delay from the root to all leaf nodes in one depth-first search. The delay time to each node can be derived from its immediate predecessor, the branch resistance, and the subtree capacitance. Recursively, in a top-down fashion we compute the delay time to each node. The outline of the recursive depth-first delay (DFD) calculation routine is as shown below.

procedure DFD(i)
1. **for each** $j \in IS(i)$ **do**
2. $t_{0j} = t_{0i} + r_j C_j$;
3. DFD(j);
4. **end for**

end procedure

Since the algorithm again runs in a depth-first search manner, the complexity is $\theta(|N|)$. A complete hierarchical delay (HD) calculation algorithm is, simply, first an execution of DFC to obtain the capacitance information, and then an execution of DFD for delay calculation of every node. The outline of HD is as follows:

procedure HD
1. DFC(0);
2. $t_{00} = 0$;
3. DFD(0);

end procedure

Since both DFC and DFD run in linear time complexity, we easily have the following theorem.

Theorem 2.1 The delay time from the root to each node on an RC tree can be computed in $\theta(|N|)$.

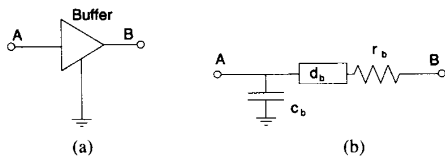

Fig. 1. (a) A clock buffer. (b) An equivalent model d_b: buffer internal delays C_b: buffer input capacitance; r_b: buffer output driving resistances; A: buffer input node; B: buffer output node.

A. Generalization to Buffered RC Trees

To handle multistaged clock trees (or buffered clock trees), we generalize the previous delay computation method for a *buffered RC tree*. Before we define what is a buffered RC tree, we first discuss a circuit model of the clock buffer as shown in Fig. 1(b). We specifically designate the input node of a buffer as a *buffer input node*, which is important for delay calculation. The box in Fig. 1 represents a delay element with d_b as the buffer internal delay and is connected to the buffer input node on one end and the buffer output driving resistor r_b on the other end. The buffer input capacitor c_b is on the buffer input node, and the buffer output driving resistor r_b is connected to the delay element and buffer output node. One function of buffers is to supply enough currents for driving latches. The other function of buffers is creating stages such that the subtree capacitance of the buffer output node will not be *carried over*, i.e., the equivalent total subtree capacitance as seen at the buffer input node is only c_b. Usually the buffer driving resistance and input capacitance are designed to be small values. This is why buffering usually reduces delay time.

To account for the buffering effects, we define a buffered RC tree to be just like a normal RC tree, except that each branch i is now also associated with a branch delay d_i besides the branch resistance r_i. The branch delay is always equal to zero, except in the case where it stands for a buffer. The basic delay calculation presented previously is modified as the following for buffered RC trees.

The calculation of the equivalent subtree capacitance at node i is now depending on whether node i is a buffer input node or not. Thus to compute the subtree capacitance of a buffered RC tree, we modify line 4 of DFC to

$$C_i = \begin{cases} C_i, & \text{if node } i \text{ is a buffer input node} \\ C_i + C_j, & \text{otherwise.} \end{cases}$$

We also extend the delay computation for a node i and its successor j as the following equation in order to accommodate the new branch delay situation, i.e.,

$$t_{ij} = \sum_{n \in N(i,j)} (r_n C_n + d_n).$$

Thus, line 2 of DFD is modified to be

$$t_{0j} = t_{0i} + r_j C_j + d_j$$

for delay calculation of a buffered RC tree, and the HD calculation routine will run exactly the same way and remain linear time complexity.

III. Delay Computation of Clock Trees

We shall discuss in this section how to model a clock tree as a buffered RC tree so that we can perform delay computation efficiently. Each clock tree realization consists of wiring segments, clock pins, and clock buffers. Hence, we shall first study the RC model of each component.

A. Equivalent π-Model for a Distributed RC Line

Distributed RC lines are more accurate for characterizing the circuit performance of wiring segments. A distributed RC line is usually represented as the symbol shown in Fig. 2(a). Either a π-model (Fig. 2(b)) or a T-model (Fig. 2(c)) is used to represent the equivalent circuit of the distributed RC line under the Elmore delay model.

Throughout this paper, we will use the equivalent π-model for the analysis. The equivalent π-model of a wire segment is represented by an input node, an output node, and a branch between both nodes. Let R be the total wire resistance and C the total wire capacitance. Then the equivalent input and output node capacitances are all equal to $C/2$, and the equivalent branch resistance is R.

B. Equivalent Buffered RC Tree of a Clock Tree

We use a generic example, as shown in Fig. 3(a), to illustrate how to construct an equivalent buffered RC tree from a multistaged clock tree. For this particular example, we assume a clock source is driving a buffer through wire 1, and the buffer is connected to the clock pin on a latch through wire 2. The driving resistance of the clock source is assumed to be r_s. Both wire segments 1 and 2 are represented by equivalent π-models as discussed earlier. The buffer is transformed to an equivalent circuit with buffer input capacitance c_b, buffer delay d_b, and buffer output driving resistance r_b. The end clock pin of the latch is associated with a loading capacitance c_l. The equivalent buffered RC tree is as shown in Fig. 3(b).

C. Lumped Delay Model

We shall introduce a *lumped delay model* based on the fact that the delay can be computed segment by segment and that the total subtree capacitance is sufficient for calculation. This model will help to ease the presentation of the zero-skew algorithm.

Recall (1):

$$t_{kj} = t_{ki} + t_{ij}.$$

Suppose i is an immediate successor of k, and j is a leaf node. Then

$$t_{kj} = d_i + r_i C_i + t_{ij}. \tag{2}$$

Consider node i as the root of the subtree T_i. To compute the delay time one level up from node k to node j, we need to know only the branch resistance r_i, the branch delay, the subtree capacitance C_i and the delay time from the root of T_i to the leaf node j, according to (1).

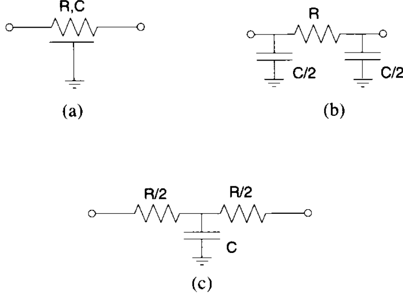

Fig. 2. (a) A distributed RC line. (b) The equivalent π-model. (c) The equivalent T-mode.

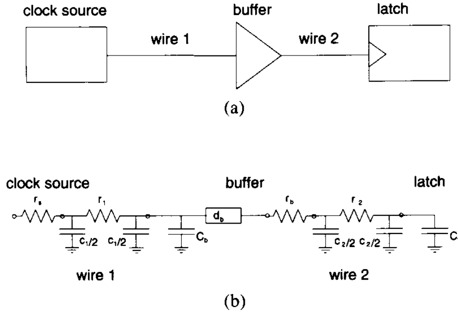

Fig. 3. (a) A generic multi-staged clock tree. (b) The equivalent buffered RC tree.

Thus we propose an equivalent lumped delay model of the subtree T_i (see Fig. 4) for simplifying the delay computation. In the equivalent circuit, the subtree T_i is replaced by an input capacitance C_i and a branch delay t_{ij} from input node i to leaf node j. We will use this lumped delay model for developing the algorithm in the next section.

IV. Zero Skew Algorithm

The zero-skew algorithm is a recursive bottom-up process. We describe only one recursive step. Repeating the process in a bottom-up fashion will construct a complete zero-skew clock tree.

We assume every subtree has achieved zero skew, which means the signal delay from the root of the subtree to its leaf nodes are equal. This is obvious if the subtree contains only one leaf node. Hence, leaf nodes are the starting subtrees of the algorithm.

To interconnect two zero-skewed subtrees with a wire and ensure zero skew of the merged tree, the problem to be solved is the decision of where on the wire the new root of the merged tree will be, such that the delay time from this new root to all leaf nodes are equal, i.e., zero skew. We will call this new *root* point on the wire a *tapping* point, and this process the *zero-skew-merge* process.

Let us discuss the example shown in Fig. 5 with two subtrees 1 and 2. First, assume the lumped delay model of each subtree is as shown in Fig. 5. The tapping point separates the interconnection wire of the two subtrees into two halves (which may not be equal). Each half wire seg-

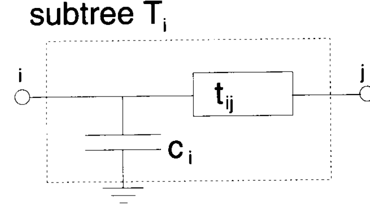

Fig. 4. An equivalent lumped delay model of a clock subtree.

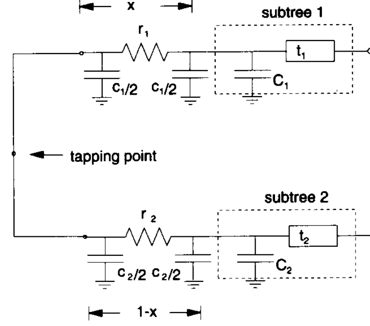

Fig. 5. *Zero-Skew Merge* of two subtrees.

ment is represented as a π-model as shown. To ensure the delay from the tapping point to leaf nodes of both subtrees being equal, it requires that

$$r_1(c_1/2 + C_1) + t_1 = r_2(c_2/2 + C_2) + t_2) \quad (3)$$

according to (2). Note that r_1 and c_1 are the total wire resistance and capacitance of the wire segment 1. Similarly, r_2 and c_2 are for wire segment 2. There are no branch delays.

We assume that the total wire length of this interconnection wire segment is l. The wire length from the tapping point to the root of subtree 1 is $x \times l$. Hence, the wire length from the tapping point to the root of subtree 2 will be $(1 - x) \times l$.

Let α be the resistance per unit length of wire and β be the capacitance per unit length of wire. Then we have $r = \alpha l$, $r_1 = \alpha x l$, $r_2 = \alpha(1 - x)l$. Also, $c = \beta l$, $c_1 = \beta x l$, $c_2 = \beta(1 - x)l$.

Hence, after solving (3), we find that the zero-skew condition requires

$$x = \frac{(t_2 - t_1) + \alpha l\left(C_2 + \frac{\beta l}{2}\right)}{\alpha l(\beta l + C_1 + C_2)}.$$

If $0 \leq x \leq 1$, the tapping point is somewhere along the segment interconnecting the two subtrees and is legal. In case that $x < 0$ or $x > 1$, it indicates the two subtrees are too much out of balance. The interconnection wire has to be elongated. For simplicity, we discuss only the case that $x < 0$. For this case, the tapping point has to be exactly on the root of subtree 1 in order to minimize total interconnection length. Assume the elongated wire length is l'. The distributed resistance value is $\alpha l'$ and the distributed capacitance value is $\beta l'$. To determine a mini-

mum elongated wire length l', it requires

$$t_1 = t_2 + \alpha l' \left(C_2 + \frac{\beta l'}{2} \right)$$

or

$$l' = \frac{[\sqrt{(\alpha C_2)^2 + 2\alpha\beta(t_1 - t_2)}] - \alpha C_2}{\alpha\beta}.$$

Similarly, for the case $x > 1$, the new root should be the root of subtree 2, and

$$l' = \frac{[\sqrt{(\alpha C_1)^2 + 2\alpha\beta(t_2 - t_1)}] - \alpha C_1}{\alpha\beta}.$$

It is worthwhile noting that the uneven loading effect is naturally taken care of by this approach.

A common practice for wire elongation is done by "snaking," as shown in Fig. 6. Since it is the nature of a clock wiring algorithm to *balance* the two subtrees, the *snaking* should not occur often. Real examples have shown less than 1.2% of wire length increase due to the elongation.

In case that the two subtrees are too much out of balance and the elongation severely affects the wirability, then addition of buffers, delay lines, or capacitive terminators should be considered, based on the same balancing principle. For instance, a capacitive terminator can be attached on the root of subtree 2 for case $x < 0$, instead of making a longer wire. The capacitance value, say C_t, can be determined by solving the equation $t_1 = t_2 + \alpha l(C_2 + \beta l/2 + C_t)$, or we will have $C_t = (t_1 - t_2)/(\alpha l) - (C_2 + \beta l/2)$.

Before presenting the algorithm formally, we define a few more related terms. The number of stages of a clock tree is defined as the maximum number of clock buffers on a path from the clock source to a clock pin, with the clock source counted as a buffer. A *cluster* is the collection of a clock buffer and its associated clock pins. Each cluster is tagged with a *stage number*, which is exactly the number of buffers on the path between the clock source and the clock buffer of the cluster. The number includes the clock source and the clock buffer of the cluster. In conclusion, we have the following efficient zero-skew clock routing algorithm.

Algorithm 4.1 (Zero-Skew Algorithm)

S1: Let s = number of clock tree stages.

S2: If s = 0, report results and exit: continue, otherwise.

S3: For each cluster in stage s, do

- *S3.1: Treat each clock pin in the cluster as a tapping point. Repeat steps S3.2 and S3.3 until there is only one tapping point left.*
- *S3.2: Pair up tapping points.*
- *S3.3: For each pair, perform zero-skew-merge of the two subtrees and determine the new tapping point, using the algorithm discussed in this section. If only one point in the group, then do nothing.*
- *S3.4: Connect the last tapping point directly to the clock driver output node.*

S4: Let s = s − 1. Continue from S2.

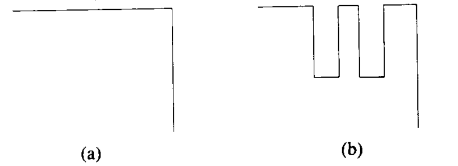

Fig. 6. (a) A regular wire. (b) Elongation of the wire by *snaking*.

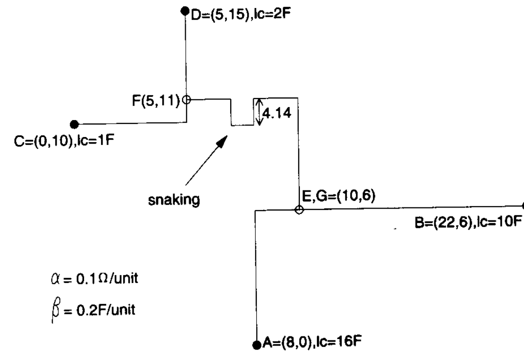

Fig. 7. A zero-skew wiring result of a simple example.

The zero-skew algorithm does not depend on the algorithm used for grouping the clock pins or tapping points into pairs. For any pairing algorithm, the zero-skew algorithm will work well. However, to optimize wirability, a minimum weighted matching algorithm may be better, or a more efficient algorithm that alternately partitions the clock pins into two equal numbered groups can be used.

When implementation occurs in real environments, we have to consider blockages and the different electric constants on different layers. The connection between any two tapping points can be done by any existing wiring algorithm that handles wiring blockages. The tapping point is then found by searching through each wiring segment of different electric constants.

To minimize the total wire length, we may construct a few possible wiring patterns (e.g., two one-bend connections) between each pair of tapping points, and pick up the one which gives shorter length at the next higher level pairing process.

Example: An example with four clock pins (Fig. 7) is used to illustrate the algorithm. Pin A is at (8, 0) with 16-F loading capacitance. Pin B is at (22, 6) with 10-F capacitance. Pin C is at (0, 10) with 1 F. Pin D is at (5, 15) with 2 F. The per unit resistance is 0.1 Ω, and the per-unit capacitance is 0.2 F. Pins A and B are in one pair and C, D in the other pair. According to the algorithm, a tapping point E is decided to be on (10, 6) so that the

TABLE I
STATISTICS OF THE TESTING EXAMPLES

Example	$R1$	$R2$	$R3$	$R4$	$R5$
No. Pins	267	598	862	1903	3101
Chip width	69 984	94 016	97 000	126 970	142 920
Chip height	70 000	93 134	98 500	126 988	145 224

TABLE II
COMPARISON BETWEEN THE ZERO-SKEW ALGORITHM AND A WIRE LENGTH BALANCING HEURISTIC

Algorithm	Zero Skew			Length Balancing	
Examples	Phase Delay (ns)	Skew (ns)	Run time(s)	Phase Delay (ns)	Skew (ps)
$R1$	1.799	0	0.1	1.798	132
$R2$	4.631	0	0.3	5.367	806
$R3$	7.055	0	0.5	7.655	702
$R4$	20.666	0	1.2	23.316	3558
$R5$	35.918	0	2.0	38.958	1931

delays to both A and B are all equal to 13.44 ns. Similarly, a tapping point F is located at (5, 11) for connection to pins C and D, with equal delay 0.96 ns. The two subtrees rooted by E and F are *very* unbalanced. We find that $x = -0.175 < 0$. Hence, the wire connecting E and F has to be elongated by 8.28 units, and the tapping point G has to coincide with E. The final wiring result is shown in Fig. 7. Note that the connections between (A, B) and (C, D) are chosen from the two one-bend connection of each pair for shorter wire length between (E, F).

V. EXPERIMENTAL RESULTS

We test our algorithm on five different sized examples. The statistics of the examples are shown in Table I. The chip width and height units are both in $1/10$ μm. We assume the per-unit resistance is 3 mΩ, and the per unit capacitance is 0.02 fF. The loading capacitances of clock pins ranged from 30 to 80 fF. For simplicity, we assume all are one-stage clock trees, i.e., no intermediate clock buffers. All experiments are conducted on an IBM 3090 machine.

We use a simple heuristic for pairing up clock pins in this experiment. We recursively partition the pins into two equal (or almost equal) halves by the median of the sorted pin list in alternate horizontal and vertical directions. This heuristic creates a binary tree for each example. Then the pins are connected, based on the zero-skew algorithm. For comparison, we also implement the wire length balancing heuristic [9] on the same binary tree. The results are shown in Table II. It is obvious that our algorithm really constructs zero-skewed clock networks according to the Elmore delay calculation. As demonstrated from the results, our algorithm shows extreme potential for cycle time improvement, especially for large chips which are becoming popular in recent products. Because of the balanced delay, the zero-skew algorithm also performs better than the wire length balancing algorithm in terms of smaller phase delay for most cases. A final clock routing result of the example $r3$ is shown in Fig. 8.

To be more convincing, we also verify the results with RICE 3.2 an accurate waveform simulation program [12]. In three test runs, we try one step input (zero rise-time ramp), and two ramp inputs with rise time 0.5 and 1.0 ns, respectively. We use 6 poles and 6 zeros for waveform simulation. The delay time at each clock pin is calculated as the difference of the time that the output is 50% of the final output value and the time that the input is 50% of the final input value. Then the skew is calculated based on the delay to each clock pin. The results are summarized in Table III. We find that the actual skew is much smaller than expected. The skew is less than 4.6 ps, even in the worst case of this experiment (see Table III).

To get better feeling on the impact of wire elongation, we also calculate the percentage of extra wire length introduced, due to the elongation, to the total wire length. As summarized in Table IV, even the worst case gives less than 1.2% penalty, which shows little effect on wirability.

VI. EXTENSIONS

In this section we outline possible extensions of the basic zero skew algorithm.

Optimum nonzero clock skew: Fishburn in [6] proposed a Linear Programming approach for an optimum clock skew assignment such that the clock period is minimized without clock hazards. This nonzero-skew assignment is mainly for the *cycle stealing* technique. The rise time of the clock signal can be adjusted to a certain time point, so that the critical paths whose delays are longer than one cycle time can be accommodated without timing violation. The result of this optimization is that the difference of the signal delays from the clock source to any two clock pins has to be a particular value. To realize the clock routing of such a nonzero-skew result requires a special routing technique to ensure an exact skew of each

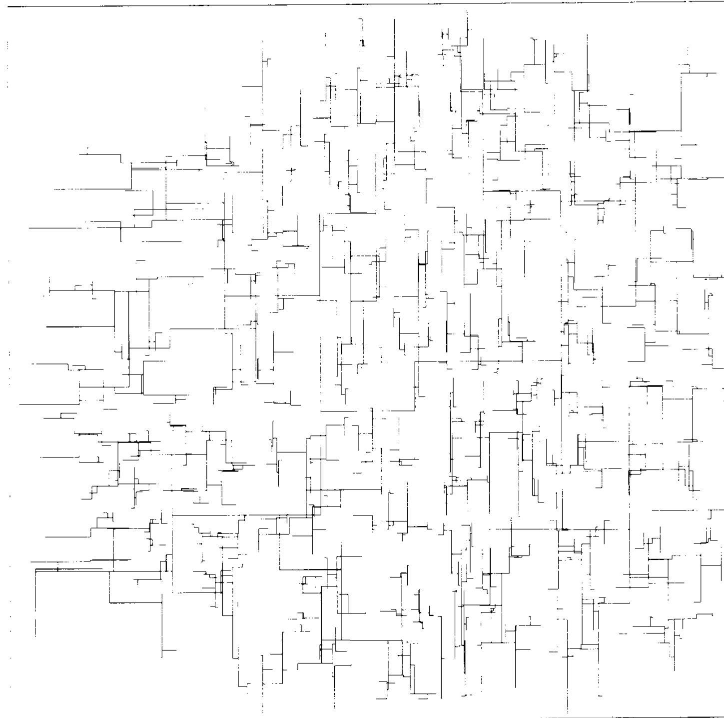

Fig. 8. A zero-skew clock routing result of the example $r3$.

TABLE III
SKEW VALUES COMPUTED FROM THE ACCURATE WAVEFORM SIMULATION PACKAGE RICE

Rise Time	0 ns	0.5 ns	1.0 ns
Examples	Skew (ps)	Skew (ps)	Skew (ps)
$R1$	0.479	0.460	0.433
$R2$	2.338	2.339	2.353
$R3$	1.304	1.304	1.302
$R4$	4.119	4.119	4.118
$R5$	4.572	4.571	4.567

TABLE IV
PERCENTAGE OF WIRE ELONGATION TO TOTAL CLOCK WIRE LENGTH

Examples	$R1$	$R2$	$R3$	$R4$	$R5$
Elongation	0.323%	0.553%	1.043%	0.415%	1.168%

pair of clock pins, as predetermined by the skew assignment algorithm.

Our zero-skew algorithm can be modified to solve this special wiring problem by adding a fictitious delay element on each clock pin. Let us assume the optimum clock delay to clock pin i is $D_0 + D_i$, where D_0 is a common offset value which is unknown until the clock routing is determined. Thus the skew between clock pin i and clock pin j is $D_i - D_j$. Let $D_{\max}$ be the maximum clock delay, i.e.,

$$D_{\max} = \max_k (D_0 + D_k) = D_0 + \max_k D_k.$$

Define the fictitious delay of clock pin i as

$$d_i = D_{\max} - (D_0 + D_i) = \max_k D_k - D_i.$$

In other words, each clock pin is modeled as a lumped delay model with an input loading capacitance and a branch delay, as shown in Fig. 9. Then we perform the zero-skew algorithm on this modified clock tree with the fictitious delay on each clock pin.

As pointed out by Fishburn [6], by combining this technique with the retiming technique [11], one can optimally make a tradeoff between chip area (in terms of number of logics and latches) and clock period.

Multi phase clock: To zero-skew a multiphase clock tree, we first perform the zero-skew algorithm on the clock subtree of each phase independently, but stop at the final end tapping point. Each subtree is then connected to the corresponding output of the clock divider by adding an appropriate delay element or snaking wires for balancing the delays according to the phase and the wiring distance to clock source (Fig. 10).

Optical or wave skew: For this case, the delay is proportional to the traveling distance. Then the zero-skew criterion is to equalize the traveling distance from the source to each end terminal. The wire length balancing algorithm [9], plus the wire elongation scheme, is perfect for this application.

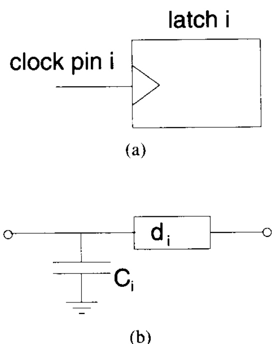

Fig. 9. (a) A clock pin on a latch. (b) The modified model of a clock pin according to the optimum skew assignment.

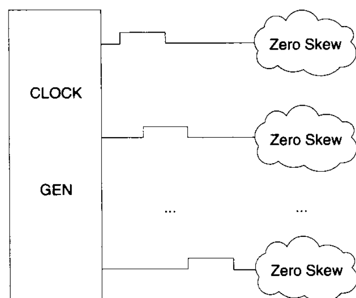

Fig. 10. Connecting a multiphase clock network.

VII. Conclusions

We have presented a novel zero-skew clock routing algorithm based on the Elmore delay calculation. The approach is ideal for hierarchical methods of constructing large systems. All subsystems can be constructed in parallel and independently, then interconnected with exact zero skew. We expect this clock routing algorithm will be widely used for performance enhancement for synchronous VLSI digital systems.

Acknowledgment

The author would like to thank Prof. Lawrence T. Pillage, Curtis L. Ratzlaff, and Nanda Gopal for their help in using the RICE package.

References

[1] H. B. Bakoglu, J. T. Walker, and J. D. Meindl, "A symmetric clock-distribution tree and optimized high speed interconnections for reduced clock skew in ULSI and WSI circuits," in *IEEE Int. Conf. Computer Design: VLSI in Computers*, 1986, pp. 118-122.

[2] P. K. Chan and K. Karplus, "Computing signal delay in general rc network by tree/link partitioning," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 898-902, Aug. 1990.

[3] T. H. Cormen, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms*. New York: McGraw-Hill, 1990.

[4] S. Dhar, M. A. Franklin, and D. F. Wann. "Reduction of clock delays in VLSI structures," in *Proc IEEE Int. Conf. Computer Design: VLSI in Computers*, pp. 1984, pp. 778-783.

[5] W. C. Elmore, "The transient response of damped linear networks with particular regard to wide band amplifiers," *J. Appl. Phys.*, vol. 19, pp. 55-63, 1948.

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

[7] A. L. Fisher and H. T. Kung, "Synchronous large systolic arrays," in *Proc. SPIE*, 1982, pp. 44-52.

[8] M. A. B. Jackson, A. Srinivasan, and E. S. Kuh, "Clock routing for high-performance IC's," in *Proc. Design Automation Conf.*, 1990, pp. 573-579.

[9] A. Kahng, J. Cong, and G. Robins, "High-performance clock routing based on recursive geometric matching," in *Proc. Design Automation Conf.*, 1991, pp. 322-327.

[10] S. Y. Kung and R. J. Gal-Ezer, "Synchronous versus asynchronous computation in very large scale integrated (VLSI) array processors," in *Proc. SPIE*. 1982, pp. 53-65.

[11] C. E. Leiserson, F. M. Rose, and J. B. Saxe, "Optimizing synchronous circuitry by retiming," in *Proc. Third Caltech Conf.*, 1983, pp. 87-116.

[12] C. L. Ratzlaff, N. Gopal, and L. T. Pillage, "RICE: Rapid interconnect circuit evaluator," in *Proc. Design Automation Conf.*, 1991, pp. 555-560.

[13] J. Rubinstein, P. Penfield, and M. A. Horowitz, "Signal delay in *rc* tree networks," *IEEE Trans. Computer-Aided Design*, vol. CAD-2 pp. 202-211, 1983.

Ren-Song Tsay (S'84-M'89) received the B.S. degree from National Taiwan University, Taiwan, Republic of China, the M.S. degree from the University of California, Santa Barbara, CA, and the Ph.D. degree from the University of California, Berkeley, CA, all in electrical engineering and computer science.

He is now with Arcsys, Inc., Santa Clara, CA. He was Research Staff Member in the Advanced Design Algorithm Group at the IBM T. J. Watson Research Center, Yorktown Heights, NY, where he has worked on several projects, including SMILE (a surface-mount packaging tool aiming at enhancement of electromagnetic compatibility), PRIZE (a high-performance circuit placement package for large chip designs), PRIDE (a timing-driven placement and routing integrated design environment), and RobinHood (a system timing verification tool specializing in cycle stealing optimization). His research interests include VLSI circuit performance optimization algorithms, early design and estimation analysis, general CAD algorithms, optimization techniques, matrix computation, and parallel processing.

Dr. Tsay received the IBM Outstanding Technical Achievement Award in 1992 and a Distinguished Paper citation at the International Conference on Computer-Aided Design (ICCAD) in 1991.