Exact Zero Skew

Ren-Song Tsay
IBM, T. J. Watson Research Center

Abstract: *An exact zero skew clock routing algorithm using Elmore delay model is presented. Recursively in a bottom-up fashion, two zero-skewed subtrees are merged into a new tree with zero skew. The algorithm can be applied to single-staged clock trees, multi-staged clock trees, and multi-chip system clock trees. It is ideal for hierarchical methods of constructing large systems. All subsystems can be constructed in parallel and independently, then interconnected with exact zero skew.*

1 Introduction

We propose in this paper 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 [2]. 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 set up, hold time, latch active time, and other possible offset factors like safety margins, for example.

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 getting larger, the clock skew is also more significant in terms of cycle time reduction.

Many heuristics have been proposed in the past for clock routing. H-tree structures are most widely used, especially in systolic array designs. A generalization of H-tree that hierarchically divides at median and connects the mean points is proposed in [3]. A further improvement is done by bottom-up pairwise connections which construct a perfect length balanced tree [4]. However, all these heuristics focus only on wire length balancing, rather than the real objective as balancing clock delay. In contrast, what we propose is an exact algorithm that directly balances the clock delays and 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 that all 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 multi-staged clock tree. Thus, the second type is called *multi-staged* clock trees that the clock pins are driven from intermediate buffers, and the buffers are driven by either other buffers or the clock source. A multi-chip system clock tree is basically a multi-staged clock tree except that the clock pins are scattered on many chips (or cards).

Then the zero skew algorithm is 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 [4].

2 Linear Time Hierarchical Delay Computation

We adopt the commonly used Elmore delay model [5] to calculate the signal traveling time from a clock source to each clock pin. We modify the method proposed in [5] and have a hierarchical method for computing delays in a bottom-up fashion, which is the key to our zero skew algorithm.

We first define some terms. Let T represent an RC tree with every node associated with an index. We assume the index of the root is always 0. A *predecessor* of node i is a node resides 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, *successors* 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

$$C_i = c_i + \sum_{k \in IS(i)} C_k \tag{1}$$

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.

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.

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

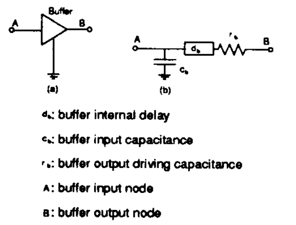
Buffer
A
B
(a)
A
d_b
r_b
B
c_b
(b)
d_b: buffer internal delay
c_b: buffer input capacitance
r_b: buffer output driving capacitance
A: buffer input node
B: buffer output node

Figure 1: (a) A clock buffer. (b) An equivalent model.

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

$$t_{kj} = t_{ki} + t_{ij} \tag{2}$$

Suppose that node k is the root (i.e. $k = 0$) and node i is the immediate predecessor of node j, then we have

$$t_{0j} = t_{0i} + r_j C_j \tag{3}$$

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

Since the time complexity of the depth first search algorithm is linear in number of edges [1] and the number of edges is the number of nodes minus one for a tree, we easily have the following theorem.

Theorem 1 *The delay time from the root to each node on an RC tree can be computed in linear time.*

Generalization to Buffered RC Trees

To handle multi-staged 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 the equivalent circuit model of a clock buffer as shown in Fig. 1b. 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 for 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 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 the case that it stands for a buffer delay. 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. Eq. (1) has to be modified as the following for computing the subtree capacitance of a buffered RC tree.

$$C_i = \begin{cases} c_i & \text{if node } i \text{ is a buffer input node} \\ c_i + \sum_{k \in IS(i)} C_k & \text{otherwise} \end{cases}$$

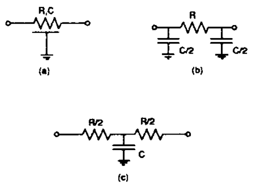

Figure 2: (a) A distributed RC line. (b) The equivalent π-model. (c) The equivalent T-model.

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, Eq. (3) is modified to be

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

for delay calculation of a buffered RC tree.

3 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 is consisted of wiring segments, clock pins, and clock buffers. Hence, it is necessary to know the equivalent RC model of each component.

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. 2a. Either a π-model (Fig. 2b) or a T-model (Fig. 2c) is used to represent the equivalent circuit of the distributed RC line.

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.

Equivalent Buffered RC Tree of a Clock Tree

We use a generic example as shown in Fig. 3a to illustrate how to construct an equivalent buffered RC tree from a multi-staged 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 the equivalent π-model 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 then shown in Fig 3b.

Lumped Delay Model

To make the presentation of the zero skew algorithm easier, we shall introduce a *lumped delay model* of a subtree. Recall Eq. (2) $t_{kj} = t_{ki} + t_{ij}$. Suppose i is an immediate successor of k, and j is a leaf node. Then

$$t_{kj} = d_i + r_i C_i + t_{ij} \tag{4}$$

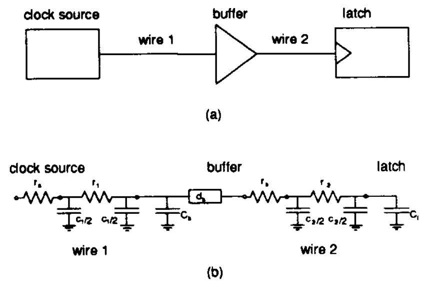

Figure 3: (a) A generic multi-staged clock tree. (b) The equivalent buffered RC tree

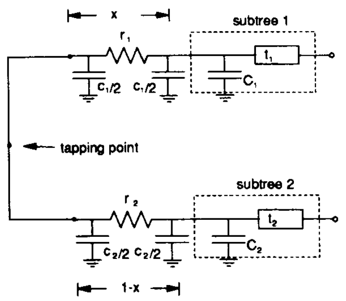

Figure 4: *Zero-Skew-Merge* of two subtrees.

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 only to know the branch resistance r_i, the branch delay d_i, the subtree capacitance C_i and the delay time t_{ij} from the root of T_i to the leaf node j according to Eq. 4.

Thus, we propose an equivalent lumped delay model of the subtree T_i 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.

4 Zero Skew Algorithm

The zero skew algorithm is a recursive bottom-up process. We describe only one recursive step. Repeat 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 delays from the root of the subtree to its leaf nodes are equal. This is obvious if the subtree contains only one leaf node, and it serves as our starting point 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 will be the new root of the merged tree, such that the delay time from this new root to all leaf nodes remain equal, i.e. zero skew. We will call this new *root* point on the wire as a *tapping* point, and this process as the *zero-skew-merge* process.

Let us discuss the example shown in Fig. 4 with two subtrees 1 and 2. First, assume the lumped delay model of each subtree is as shown in Fig. 4. The tapping point separate the interconnection wire of the two subtrees into two halves (which may not be equal). Each half wire segment is represented by π-model as shown. To ensure the delay from the tapping point to leaf nodes of both subtrees being equal, it requires that

$$r_1(c_1/2 + C_1) + t_1 = r_2(c_2/2 + C_2) + t_2 \qquad (5)$$

according to Eq. (4). 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 Eq. (5), we find that the zero skew condition requires

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

If $0 \le x \le 1$, the tapping point is somewhere along the segment interconnecting the two subtrees and is legal. In case that $x < 0$ or $x > 1$, no tapping point can be found on the interconnecting segment. It has to be elongated. For simplicity, we discuss only the case that $x < 0$. For this case, the tapping point should 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 minimum elongated wire length l', it requires

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

or

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

Similar results can be obtained for the case $x > 1$. It is worth while noting that the uneven loading effect is naturally taken care of by this approach.

A common practice for wire elongation is done by "snaking". Since it is the nature of a clock wiring algorithm to *balance* the two subtrees, the *snaking* should not occur often.

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, the capacitance value, say C_t, of a capacitive terminator to be attached on the root of subtree 2 for case $x < 0$ can be determined by solving the equation $t_1 = t_2 + \alpha l(C_2 + \beta l/2 + C_t)$, or $C_t = (t_1 - t_2)/(\alpha l) - (C_2 + \beta l/2)$.

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. Then we have the following zero skew clock routing algorithm.

Algorithm 1 (Zero Skew Algorithm)

S1: Let s = number of clock tree stages.

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

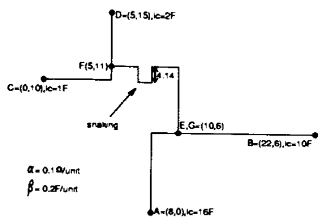

Figure 5: A zero skew wiring result of a simple example.

```
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
          buffer output node.
S4: Let $s = s - 1$. Continue from S2.
```

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 always work. However, to optimize wirability, minimum weighted matching algorithm maybe better for pairing. Or a more efficient algorithm that alternately partitions the clock pins into two equal-numbered groups can be used.

When implementation 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 (ex. 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. 5) is used to illustrate the algorithm. Pin A is at (8,0) with 16F loading capacitance. Pin B is at (22,6) with 10F capacitance. Pin C is at (0,10) with 1F. Pin D is at (5,15) with 2F. The per unit resistance is 0.1Ω, and the per unit capacitance is 0.2F. Pin A and B are in one pair and C, D in other pair. According to the algorithm, a tapping point E is decided to be on (10,6) so that the delays to both A and B are all equal to 13.44ns. Similarly, a tapping point F is located at (5,11) for connection to pins C and D, with equal delay 0.96ns. The two subtrees rooted by E and F are VERY unbalanced. We find that $x = -0.175 < 0$. 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. 5. Note that the connections between (A,B) and (C,D) are chosen from the two one-bend connections of each pair for shorter wire length between (E,F).

5 Experimental Results

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

Examples	r1	r2	r3	r4	r5
No. Pins	267	598	862	1903	3101
chip width	69984	94016	97000	126970	142920
chip height	70000	93134	98500	126988	145224

Table 1: The statistics of the testing examples.

Algorithm	Zero Skew			Length Balancing	
Examples	phase delay(ns)	skew (ns)	runtime (s)	phase delay(ns)	skew (ns)
r1	1.799	0	0.1	1.798	0.132
r2	4.631	0	0.3	5.367	0.806
r3	7.055	0	0.5	7.655	0.702
r4	20.666	0	1.2	23.316	3.558
r5	35.918	0	2.0	38.958	1.931

Table 2: A comparison between the zero skew algorithm and a wire length balancing heuristic.

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 [4] on the same binary tree. The results are shown in Table 2. It is needless to say that the zero skew algorithm is very important for eliminating the clock skew, especially as for large chips.

6 Conclusions

We have presented a novel zero skew clock routing algorithm based on Elmore delay calculation. The approach is ideal for constructing large systems. It can also be modified for customized skew (for cycle stealing) and muti-phase clock systems [6]. We expect this clock routing algorithm will be widely used for performance enhancement for synchronous VLSI digital systems.

References

[1] T. H. Corman, C. E. Leiserson, and R. L. Rivest. *Introduction to Algorithms.* McGraw Hill, New York, 1990.

[2] J. P. Fishburn. Clock skew optimization. *IEEE Transactions on Computer*, C-39(7):945–951, 1990.

[3] M. A. B. Jackson, A. Srinivasan, and E. S. Kuh. Clock routing for high-performance IC's. In *Proceedings of Design Automation Conference*, pages 573–579, 1990.

[4] Andrew Kahng, Jason Cong, and Gabriel Robins. High-performance clock routing based on recursive geometric matching. In *Proceedings of Design Automation Conference*, pages 322-327, 1991.

[5] J. Rubinstein, P. Penfield, and M.A. Horowitz. Signal delay in rc tree networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2(3):202-211, 1983.

[6] Ren-Song Tsay. Exact zero skew. Technical Report RC 16683, IBM Yorktown Research Center, 1991.