Computing Signal Delay in General RC Networks by Tree/Link Partitioning

Pak K. Chan and Kevin Karplus
Computer Engineering
University of California, Santa Cruz
Santa Cruz, California 95064

Abstract

Most *RC* simulators only handle tree networks, not arbitrary networks. We present an algorithm for computing signal delays in general RC networks using the RC-tree computation as the primary operation. We partition a given network into a spanning tree and link branches. Then we compute the signal delay of the spanning tree, and update the signal delay as we incrementally add the links back to reconstruct the original network. If m is the number of link branches, this algorithm requires $\frac{m(m+1)}{2}$ updates and $m+1$ tree delay evaluations. All the tree delay evaluations involve computing signal delays with the same resistive spanning tree, but with different values for the capacitors.

1 Introduction

The linear *RC* model has become an acceptable and pragmatic approach for modeling digital MOS circuits in the past decade. Research has been carried out both in bounding the waveforms [RPH83, Wya85, Zuk86] and in estimating the signal delays [BNR*87, CS89, LM84, Ous85, Ter83] of *RC* networks. In particular, Elmore's notion of signal delay [Elm48] has been used widely to approximate the time taken for a signal to start from an initial value and reach half of its final value. If $\mathbf{G}$ is the node-conductance matrix of a given *RC* network and $\mathbf{C}$ is the capacitance matrix of the network, calculating Elmore's delay, $\mathbf{t}_d$, can be as simple as evaluating the product of $\mathbf{G}^{-1}$, $\mathbf{C}$, and unit vector $\mathbf{1}$. Since $\mathbf{G}$ and $\mathbf{C}$ are given, the delay estimation problem amounts to computing the resistance matrix, $\mathbf{R} \equiv \mathbf{G}^{-1}$. Thus delay estimation in *RC* networks has been viewed as a numerical problem: inverting $\mathbf{G}$ [LM84]. The computational requirement of this numerical approach limits its applicability to general problems. However, if the *RC* network is a tree, then $\mathbf{R}$ can be determined by inspection, and $\mathbf{t}_d$ can be computed in linear time. Almost all MOS timing-level simulators treat networks as if they were trees, trading off accuracy for simplicity [BNR*87, Ous85, Ter83].

Signal delays in tree networks are easy to evaluate; unfortunately, many practical MOS circuits aren't trees. The Manchester adder with carry-bypass circuitry, as depicted in Fig. 1, is an example [WE85]. Since the bypass transistor B is connected to $\overline{C}_0$ and $\overline{C}_4$, when this transistor is ON and all the P_i are high, they form a closed loop of conducting transistors, and cannot be modeled as an *RC* tree.

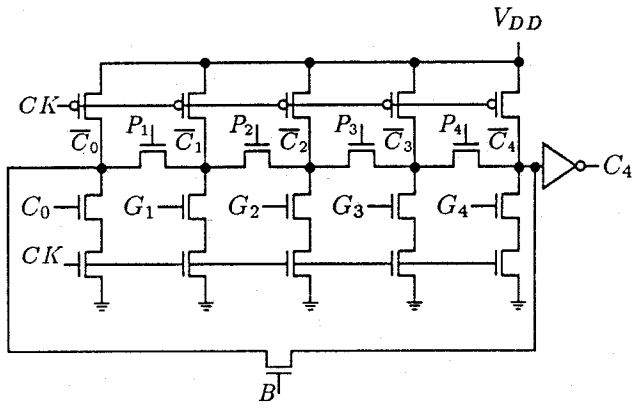


Figure 1: Manchester Adder with Carry-Bypass

The goal of this work is to provide an efficient way of computing signal delays in *RC* networks that do not necessarily form trees. We show that based on the idea of Kron's branch tearing, we can partition a given network into a spanning tree and link branches [Kro39]. We then start with the signal delay of the *spanning tree*, and gradually update the signal

delay while incrementally piecing the *links* back to reconstruct the original network. The important concept of Kron's tearing will be explained in Section 3. A delay computation algorithm and its complexity will be explained in Section 4. Here, we shall only consider RC networks with grounded capacitors at each node of the network and no floating capacitors. Extensions to handle floating capacitors are discussed in [Cha88].

A note on related work: Lin and Mead invented the technique of "tree decomposition" and "load redistribution" to calculate signal delay in general RC networks [LM84]. Their algorithm is relaxation-based, and the number of relaxation steps depends on the required accuracy. The algorithm that we are presenting here is based on dynamic programming, and terminates in a predetermined number of steps with the exact result.

2 Signal Delay in RC Networks

The *delay estimation problem* aims to find the time interval that it would take a signal to start from an initial value and reach a prescribed value. The most meaningful such value for digital circuits is the threshold voltage where the two logic states cross. However, locating this delay time precisely can be as hard as finding the exact waveform. Many researchers have used the notion of delay as defined by Elmore [Elm48] to approximate such a delay time [Ash64, BNR*87, Cha86, LM84, Zak72]. Elmore suggested defining delay as the *normalized* first moment of the impulse response $h(t)$:

$$t_d \equiv \frac{\int_{0^+}^{\infty} h(t)tdt}{v(\infty) - v(0^+)} \tag{1}$$

where $v(\infty)$ and $v(0^+)$ are the final and initial voltages. If $v(t)$ is the voltage response due to a unit step input, then an equivalent definition of Elmore's delay is [BNR*87, Cha86]

$$t_d \equiv \frac{\int_{0^+}^{\infty}[v(\infty) - v(t)]dt}{v(\infty) - v(0^+)}. \tag{2}$$

Based on this notion of signal delay, closed-form delay expressions can be derived for RC networks without floating capacitors. We note that for a simple RC circuit, Elmore's delay is the same as the time constant of the circuit.

For a given RC network with n grounded capacitors, let

- $G_{i,j}$ be the branch conductance between nodes i and j (and by reciprocity, $G_{i,j} = G_{j,i}$),
- $G_{i,i}$ be the sum of all branch conductances connected to node i, and
- C_i denote the capacitance at node i.

The node-conductance matrix of the network has the following form:

$$\mathbf{G} = \begin{bmatrix} G_{1,1} & -G_{1,2} & \cdots & -G_{1,n} \\ -G_{1,2} & G_{2,2} & & -G_{2,n} \\ \vdots & & & \vdots \\ -G_{1,n} & -G_{2,n} & \cdots & G_{n,n} \end{bmatrix}.$$

With this, Elmore's delay for the i^{th} node in the RC network is

$$t_{d,i} \equiv \frac{\sum_{j=1}^{n} R_{i,j}C_j[v_j(\infty) - v_j(0^+)]}{v_i(\infty) - v_i(0^+)} \tag{3}$$

or in matrix form

$$\mathbf{t}_d \equiv \frac{\mathbf{RC}[\mathbf{v}(\infty) - \mathbf{v}(0^+)]}{\mathbf{v}(\infty) - \mathbf{v}(0^+)} \tag{4}$$

where $\mathbf{R} \equiv \mathbf{G}^{-1}$, $\mathbf{C}$ is a diagonal matrix with entries $(C_1, C_2, ..., C_n)$, $\mathbf{v}(\infty)$ and $\mathbf{v}(0^+)$ are the final and initial node voltages. For proof, see [Cha86, LM84].

Speaking in terms of the components of the $\mathbf{R} = (R_{i,j})$ matrix, $R_{i,i}$ is the driving-point resistance between node i and the input node, and $R_{i,j}$ is the *transfer resistance* between nodes i and j. If the given RC network is a tree, then it has only one spanning tree and $R_{i,j}$ is the resistance of the portion of the unique path between the input and the j^{th} node that is common with the unique path between the input and node i. In particular, $R_{i,i}$ is the resistance between the input and node i [RPH83]. Hence, the resistance matrix $\mathbf{R}$ can readily be found by inspection. Evaluating delays for all n nodes of a tree network requires only n multiplications.

We shall focus on treatments of RC networks which are not necessarily trees for the rest of the article.

3 Circuit Partitioning

Tearing is a means of partitioning circuits into several smaller, more manageable subcircuits to enhance computational efficiency. A large circuit is "torn" into simpler subcircuits. We solve the subcircuits, afterward we piece together the subcircuit solutions with a formal mechanism to yield the composite result.

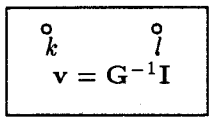
We need some notations. An RC circuit is abstracted by a circuit graph $\mathcal{G} = (N, E)$, consisting of a set of nodes $N = \{n_0, n_1, \ldots, n_n\}$ and a set of edges $E = \{e_1, \ldots, e_b\}$. We assume that each node in the RC circuit has a grounded capacitor, and a resistor associated with each edge e_i. Without loss of generality, the node n_0 designates the input node. Furthermore, we can partition $\mathcal{G}$ into two parts: a spanning tree consisting of n edges, and $b - n$ edges that do not belong to the tree, called link branches.

3.1 Tree/Link Partitioning

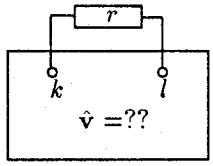
Given a network, we can partition it into a spanning tree and link branches using a simple depth-first search of the graph. We start by solving the spanning tree, and update the solution by incrementally adding the links back to reconstruct the original network. The mechanism that we use for updating the solution is based on a formula derived by Rohrer, which shows the consequence of adding a resistor to a circuit that has been solved [Roh88]. Adding more than one circuit element can be treated inductively by adding one element at a time. The idea of Tree/Link partitioning has previously been applied to a piecewise linear circuit simulator [HPR87, PHR87], and to solving linear equations by tree relaxation [SZ88].

3.2 Adding a Single Resistor

Suppose that we have already computed all the node voltages $\mathbf{v}$ of a resistive circuit $\mathbf{G}$ driven by the current source vector $\mathbf{I}$ (Fig 2.a), and we wish to ascertain the consequences of adding a resistor of value r between its k^{th} and l^{th} nodes, as illustrated in Fig 2.b.



(a)



(b)

Figure 2: Updating a Resistive Circuit

Rohrer [Roh88] shows that the effect on the circuit node voltages of the addition of a resistor r between nodes k and l is

$$\hat{\mathbf{v}} = \mathbf{v} - \frac{v_k - v_l}{r + R_{k,k} + R_{l,l} - 2R_{k,l}} \mathbf{R}\xi_{k,l}, \qquad (5)$$

where $\xi_{k,l}$ is a column vector with a "+1" in the k^{th} row, a "−1" in the l^{th} row, and zeros everywhere else. We note that the connection vector $\xi_{k,l}$ indicates that the resistor to be added is connected from node k to node l.

3.3 Applying Tree/Link Partitioning to Signal Delay Estimation

First we consider networks with only one driving-source (either V_{DD} or ground), so that, all node voltages $v_i(\infty)$ attain the same final value. The effect of multiple driving-sources and different final node voltages will be considered in Section 3.4. Assuming that all initial node voltages are the same, we apply Elmore's definition of delay to equation (5) to obtain the following formula for updating delay values:

$$\hat{\mathbf{t}}_d = \mathbf{t}_d - \frac{t_{d,k} - t_{d,l}}{r + R_{k,k} + R_{l,l} - 2R_{k,l}} \mathbf{R}\xi_{k,l}. \qquad (6)$$

The strategy for computing the signal delay for arbitrary RC networks is clear: we remove all the link branches until we have a spanning tree, compute the signal delay of the tree, and then gradually add back the previously deleted link branches using equation (6).

This approach has the following desirable properties: first, the order of the removal of the link branches is arbitrary, which makes implementation easier. Second, it is not necessary to compute the signal delay of all nodes: the computation involved can be limited to the nodes for which signal delays are required - for instance, the primary output nodes. Third, the complexity of this approach depends on the number of link branches. Since VLSI circuits are nearly trees, this approach lends itself well to VLSI applications.

Existing switch-level simulators handle tree networks very effectively. To facilitate the incorporation of our idea into these simulators, we shall formulate our method using the tree delay evaluation as the primary operation. Referring to equation (6), we note that $\mathbf{R}\xi_{k,l}$ can be conceived of as a delay calculation with the grounded capacitances of the k^{th} and l^{th} nodes set to +1 and −1 respectively, and the rest set to zeros. Furthermore, if we define $\mathbf{r} = (r_i) \equiv \mathbf{R}\xi_{k,l}$, then the delay expression can be reformulated as

$$\hat{\mathbf{t}}_d = \mathbf{t}_d - \frac{t_{d,k} - t_{d,l}}{r + r_k - r_l} \mathbf{r}. \qquad (7)$$

This formula, and the algorithm to be presented in Section 4, constitute the major result of this article.

Example: Fig. 3 shows an instance of the RC model for the transistor loop in the Manchester adder as shown in Fig. 1. With the switch closed, deleting any one of the resistors $R_1, ..., R_5$ from the circuit will result in a tree, so we arbitrarily remove R_3 and compute the signal delay of the remaining spanning tree, as shown in Fig. 4.a.

$$\mathbf{t}_d = \begin{bmatrix} R_6(\sum_{i=1}^{5} C_i) \\ R_6(\sum_{i=1}^{5} C_i) + R_1(C_2 + C_3) \\ R_6(\sum_{i=1}^{5} C_i) + R_1(C_2 + C_3) + R_2 C_3 \\ R_6(\sum_{i=1}^{5} C_i) + R_5(C_4 + C_5) + R_4 C_4 \\ R_6(\sum_{i=1}^{5} C_i) + R_5(C_4 + C_5) \end{bmatrix} = \begin{bmatrix} 1000 \\ 1200 \\ 1300 \\ 1500 \\ 1400 \end{bmatrix}.$$

Let's call this an RC tree delay operation. Next, we set C_3 to +1, C_4 to −1, and the rest of the capacitors

to zeros, as shown in Fig. 4.b, to compute $\mathbf{r}$. This gives us $\mathbf{r} = \begin{bmatrix} 0 \\ R_1 \\ R_1+R_2 \\ -(R_4+R_5) \\ -R_5 \end{bmatrix} = \begin{bmatrix} 0 \\ 10 \\ 20 \\ -30 \\ -20 \end{bmatrix}$. Now, we *merge* the results of the two tree evaluations using equation (7) to obtain the signal delay of the original network:

$$\hat{\mathbf{t}}_d = \mathbf{t}_d - \frac{t_{d,3} - t_{d,4}}{R_3 + r_3 - r_4}\mathbf{r} = \begin{bmatrix} 1000.0 \\ 1233.3 \\ 1366.7 \\ 1400.0 \\ 1333.3 \end{bmatrix}.$$

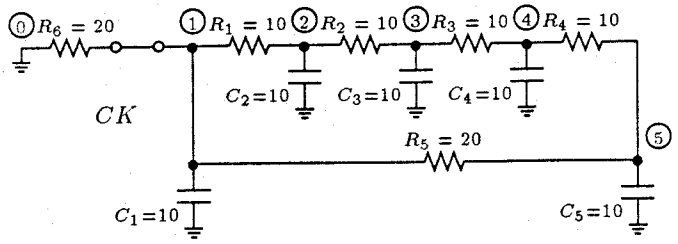


Figure 3: Manchester Adder RC Model

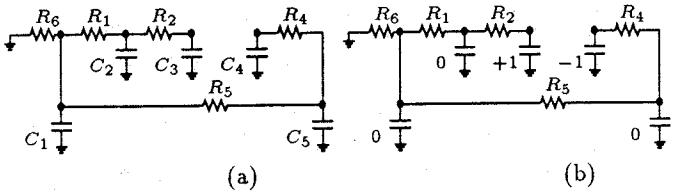


Figure 4: Two Tree Evaluations as a result of Partitioning

For a connected network with $n+1$ nodes and n edges, only one tree delay evaluation is needed. We have just shown that it takes two tree delay evaluations to compute the signal delay of the carry-bypass circuit with a single loop. In general, it takes $b-n+1$ tree delay evaluations for a network consisting of $n+1$ nodes, b edges, and $b-n$ link branches. A general analysis will be given in the Section 4.

3.4 Networks with Two Driving Sources

To compute signal delays in a network with two driving sources, we need to know the final node voltages. We compute them by using Kron's branch tearing: we partition the network into subnetworks, each driven by a single source. We use equation (5) to compute the final node voltages (with the renaming $\mathbf{r} = \mathbf{R}\xi_{k,l}$), namely:

$$\hat{\mathbf{v}}(\infty) = \mathbf{v}(\infty) - \frac{v_k(\infty) - v_l(\infty)}{r + r_k - r_l}\mathbf{r}. \qquad (8)$$

The effect of different final and initial node voltages on signal delays can be easily accommodated by the following trick. Referring to equation (3), imagining that each node capacitance is of value $Q_j = C_j[\hat{v}_j(\infty) - \hat{v}_j(0^+)]$, we compute the "signal delays" of each subnetwork with these "capacitances". The "signal delay" values computed are merged by using equation (7); then are divided by $\hat{v}_i(\infty) - \hat{v}_i(0^+)$ to obtain the actual signal delay. The following simple example will illustrate the idea.

Fig. 5.a shows an RC network with a leakage path to ground. Let $V(\infty) = 1$ volt, and $v_1(0^+) = v_2(0^+) = 0$ volt. The signal delay of this network can be computed by considering the subnetworks (forest) shown in Fig. 5.b and 5.c, obtained by deleting R_2 from Fig. 5.a. First, we obtain $\mathbf{r}$ by considering the subnetworks shown in Fig. 5.c: $\mathbf{r} = \begin{bmatrix} R_1 \\ -R_3 \end{bmatrix}$. By equation (8), the final node voltages are

$$\begin{aligned} \hat{\mathbf{v}}(\infty) &= \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \frac{1}{R_2 + R_1 - (-R_3)} \begin{bmatrix} R_1 \\ -R_3 \end{bmatrix} \\ &= \frac{1}{R_1 + R_2 + R_3} \begin{bmatrix} R_2 + R_3 \\ R_3 \end{bmatrix}. \end{aligned}$$

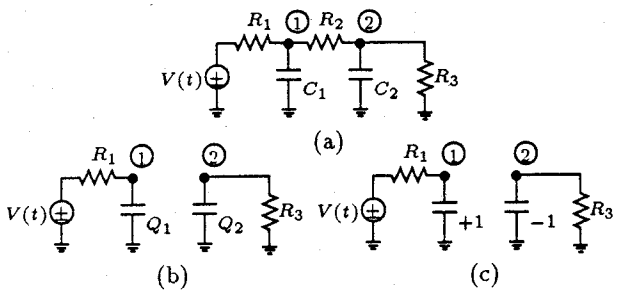


Figure 5: Tree Evaluations as a result of Partitioning

Next we compute the "signal delays" of Fig. 5.b, which yields $\mathbf{t}_d = \begin{bmatrix} R_1C_1\hat{v}_1(\infty) \\ R_3C_2\hat{v}_2(\infty) \end{bmatrix}$. Finally, we use equation (7) $\hat{\mathbf{t}}_d = \frac{\mathbf{t}_d - \frac{t_{d,1}-t_{d,2}}{R_2+r_1-r_2}\mathbf{r}}{\hat{\mathbf{v}}(\infty)}$ to obtain the result

$$\frac{1}{R_1+R_2+R_3} \begin{bmatrix} R_1(R_2+R_3)C_1 + R_1\frac{R_3^2}{R_2+R_3}C_2 \\ R_1(R_2+R_3)C_1 + R_3(R_1+R_2)C_2 \end{bmatrix}.$$

4 Complexity

Let $n+1$ be the number of nodes and b be the number of edges in a given network. Then $m \equiv n - b$ is the number of link branches. We show that the algorithm uses $m+1$ tree evaluations and $\frac{m(m+1)}{2}$ merges. Notice that the update operation in equation (7) can be applied only to two circuits with the same structure $\mathbf{R}$. After the update, the new delays correspond to a different resistance structure. To add another link branch to the circuit, we first have to compute the

delay for that branch with the modified resistance structure.

Given a network with circuit graph $\mathcal{G} = (N, E)$, let:

- $s \leq n$ be the number of specific nodes for which signal delays are required,
- $L = \{b_1, \cdots, b_m\}$ be any set of link branches, and (p_j, q_j) be the nodes to which the link branch b_j is connected,
- $X(i)$ be a circuit with the original capacitors but with link branches b_{i+1} to b_m deleted from $\mathcal{G}$, and
- $Z(i,j)$ be a contrived circuit with links b_{i+1} to b_m deleted from $\mathcal{G}$, and with the $+1$ and -1 capacitors connected to nodes p_j and q_j respectively, and the rest of capacitors zeroed; defined only for $j > i$.

The $m+1$ tree evaluations are performed on $X(0)$ (the spanning tree with the original capacitors), and $\{Z(0,j);\ j = 1, \ldots, m\}$ (the spanning trees with the $+1$ and -1 capacitors connected to nodes p_j and q_j). Each tree evaluation requires n operations.

Starting with the trees $X(0), Z(0,1), ..., Z(0,m)$, the algorithm for combining the circuits can be expressed by the recurrences:

$$\begin{cases} Z(i,j) = \bigotimes(Z(i-1,i), Z(i-1,j)) & i = 1, \cdots, m\ ;\ j > i \\ X(i) = \bigotimes(X(i-1), Z(i-1,i)) & i = 1, \cdots, m \end{cases}$$

where $\bigotimes$ denotes the operation of computing delay using equation (7); with the first argument providing $\mathbf{t}_d$ and the second providing $\mathbf{r}$. Because the *merge* operation that involves link branch b_j needs the signal delays of nodes (p_j, q_j), the *merge* operation takes $O(m+s)$ additions and multiplications. Overall, the time complexity of the algorithm is $O(n(m+1)+m^3+sm^2)$, and the space complexity is $O(m^2+sm)$.

The structure of the computation is illustrated in Fig. 6, clearly showing the $\frac{m(m+1)}{2}$ *merges* that are required. Each triangle denotes a tree delay evaluation of the labelled circuit, and each diamond denotes a *merge* operation using equation (7). We reiterate the important point that all $m+1$ tree delay evaluations use the same resistive spanning tree, but with different values for the capacitors.

If $O(n(m+1)+m^3+sm^2)$ operations are too many for some applications, faster approximate answers can be obtained by restoring only a few of the link branches. The circuits $X(0), \cdots, X(m-1)$ can be considered as approximations to $X(m)$. Premature termination of the algorithm will deliver the signal delay of an approximated circuit.

If $\mathcal{G}$ is a connected simple planar graph, the maximum number of edges is $3(n+1)-6$ [Wil79]. Therefore, the time complexity of the algorithm for a planar circuit is $O(n^3)$. The performance on real circuits is expected to be much better, nearly linear in n, since VLSI circuits are tree-dominant.

5 Remarks and Conclusions

We have presented a simple technique for computing delays in arbitrary networks of resistors with grounded capacitors. The technique allows rapid computation for *RC* networks that are almost trees, it is therefore particularly appropriate for MOS timing simulators.

We have found applications of this technique to other areas – for example, calculating node voltages and branch currents of a power supply network. Given a resistance network which models the power supply network, and current sources at each node, the problem is to calculate all the node voltages. We can do arbitrary tree/link partitioning on the power supply network and apply equation (5) directly. Previous work has provided only approximate solutions to the problem. For example, `Ariel` is a tool that attempts to solve the problem by splitting up the resistance network into a set of trees [SH88]. It is possible that `Ariel` may not split the network at the proper position and node voltage estimations will be overly conservative.

We are currently expanding the work to allow true incremental computation of delays as resistances change, for still more efficient evaluation in MOS timing simulators. We shall implement the resulting technique in an existing timing simulator, probably `TIMEMILL` [TIM88].

References

[Ash64] K. G. Ashar. The Method of Estimating Delay in Switching Circuits and the Figure of Merit of a Switching Transistor. *IEEE Trans. on ED*, 11(7):497–506, Nov. 1964.

[BNR*87] R. Bauer, P.-C. Ng, A. Raghunathan, M.W. Saake, and C.D. Thompson. Simulating MOS VLSI circuits using Supercrystal. In *Intl. Conf. VLSI 87, IFIP*, 1987.

[Cha86] Pak K. Chan. An Extension of Elmore's Delay. *IEEE Trans. on CAS*, CAS-33(11):1147–1149, Nov. 1986.

[Cha88] Pak K. Chan. Signal Delay in *RC* Networks with Floating Capacitors. In *Proc. of Intl. Symp. on Circuits and Systems*, Finland, June 1988.

[CS89] Pak K. Chan and Martine D.F. Schlag. Bounds on Signal Delay in *RC* Mesh Networks. To appear in *IEEE Trans. on CAD*, May 1989.

[Elm48] W. C. Elmore. The Transient Response of Damped Linear Networks with Particular Regard to Wideband Amplifiers. *J. Appl. Phys.*, 19(1):55–63, Jan. 1948.

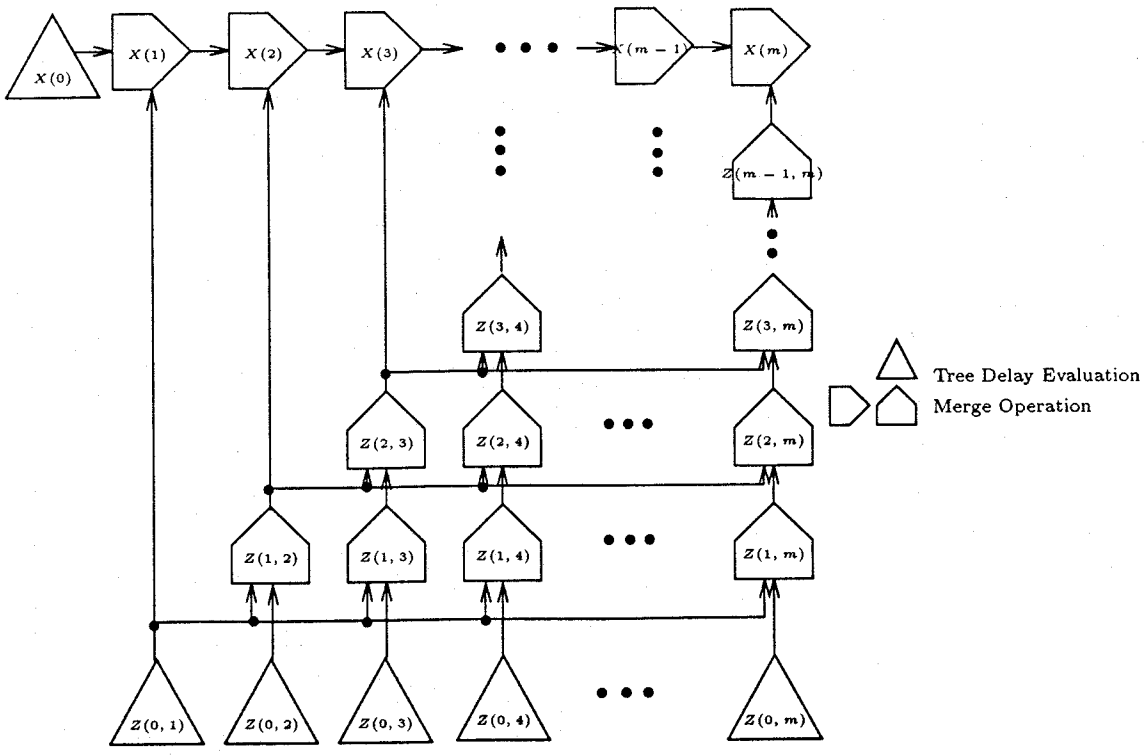


Figure 6: Combining Subcircuits

[HPR87] X. Huang, L.T. Pillage, and R. Rohrer. TALISMAN: A Piecewise Linear Simulator Based on Tree/Link Repartitioning. In *IEEE Intl. Conf. on CAD ICCAD-87*, pages 98–101, Nov. 1987.

[Kro39] G. Kron. *Tensor Analysis of Networks.* New York: Wiley, 1939.

[LM84] T.-M. Lin and C. A. Mead. Signal Delay in General *RC* Networks. *IEEE Trans. on CAD*, CAD-3(4):331–349, Oct. 1984.

[Ous85] J. K. Ousterhout. A Switch-Level Timing Verifier for Digital MOS VLSI. *IEEE Trans. on CAD*, CAD-4(3):336–348, July 1985.

[PHR87] L.T. Pillage, X. Huang, and R. Rohrer. Tree/Link Partitioning for the Implicit Solution of Circuit Equations. In *Proc. of Intl. Symp. on CAS*, pages 1072–1075, June 1987.

[Roh88] R. A. Rohrer. Circuit Partitioning Simplified. *IEEE Trans. on CAS*, 35(1):2–5, January 1988.

[RPH83] J. Rubinstein, P. Penfield Jr., and M. Horowitz. Signal Delay in *RC* Tree Networks. *IEEE Trans. on CAD*, CAD-2(3):202–211, July 1983.

[SH88] D. Stark and M. Horowitz. Analyzing CMOS Power Supply Networks using Ariel. In *ACM IEEE 25th DAC Proc.*, pp. 460–464, June 1988.

[SZ88] C. Shi and K. Zhang. Tree Relaxation: A New Iterative Solution Method for Linear Equations. In *Proc. of Intl. Symp. on CAS*, pages 2355–2358, Finland, June 1988.

[TIM88] *TIMEMILL User Manual.* EPIC Design Technology, 2900 Lakeside Drive, Santa Clara, CA 95050, 1988.

[Ter83] C. J. Terman. *Simulation Tools for Digital LSI Design.* PhD thesis, MIT, Sept. 1983.

[WE85] N. Weste and K. Eshraghian. *Principles of cMOS VLSI Design—a Systems Perspective.* Addison-Wesley, 1985.

[Wil79] R.J. Wilson. *Introduction to Graph Theory.* Academic Press, 1979.

[Wya85] John L. Wyatt, Jr. Signal Delay in *RC* Mesh Networks. *IEEE Trans. on CAS*, CAS-32(5):507–510, May 1985.

[Zak72] R. A. Zakarevicius. The Time Delay of a Network: Characterization and Measurement. *Proc. of the IREE*, 556–560, Dec. 1972.

[Zuk86] C.A. Zukowski. Relaxing Bounds for Linear *RC* Mesh Circuits. *IEEE Trans. on CAD*, CAD-5(2):305–312, April 1986.