

# Switch-Level Simulation Using Dynamic Graph Algorithms

Dan Adler

**Abstract**—A new model for MOS transistors suitable for logic simulation of VLSI circuits is presented, based on the concept of a dynamically directed switch (DDS). As in other switch-level models, the circuit is viewed as a graph where on-transistors are represented by directed edges, and circuit nodes are represented by vertices. The problem of finding the steady-state response of the circuit is shown to be reducible to the single-source shortest path problem in graph theory. A distributed iteration process is defined where each transistor determines its instantaneous direction based on its source and drain vertex labels, and updates those labels according to its own resistance. Two types of events in the circuit require the re-evaluation of the graph labels: transistors turning on and off. These correspond to edge addition and deletion in the circuit graph. Rather than repeatedly applying a shortest path algorithm to the graph whenever these events occur, we investigate the possibility of dynamically updating only those labels which were affected by the change. The main result of this paper is the development of an algorithm for switch-level simulation based on this incremental-updating technique using only local information. The resulting algorithm is fast and accurate, and has been extended to deal with timing based on an RC-tree delay model. The implementation of the algorithms presented here within the *Lsim Mixed-Mode Analog and Digital Simulator* is described.

## I. INTRODUCTION

OVER THE past several years, switch-level simulation has grown in popularity as an effective method for logic-level verification of MOS VLSI designs. Seeking to improve the unidirectional transfer gate model offered by most logic simulators, switch-level simulators have resorted to more complex algorithms in order to find the steady-state response of a digital network. The tradeoff, as always, is speed versus accuracy. The most commonly cited advantages of the switch-level model over gate-level simulators are their ability to model bidirectional signal flow, static charge sharing, radioed logic, and unknown logic states.

The demand for high-speed switch-level simulators has led to the development of numerous algorithms. Although it is possible to classify these algorithms in different ways [1], in this paper we will mainly be concerned with the distinction of *global* versus *distributed* algorithms (see Fig. 1). In global algorithms, a solution is achieved by simultaneously solving a set of equations, or by tracing paths in a graph representing part of the network. Distributed algorithms, on the other hand, aim to achieve the same result by considering only a single component at a time, and its interaction with its neighboring components. The distributed approach to switch-level simulation is the preferred method in extended gate-level simulators, where the same *event scheduling* mechanism is used to deal with both bidirectional

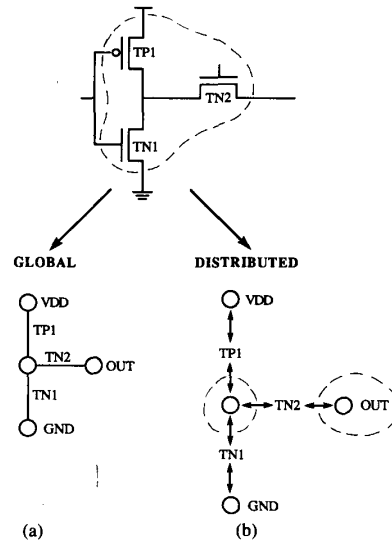


Fig. 1. (a) Global view of channel graph. (b) Distributed view.

and unidirectional elements. The model most commonly used is Hayes' model [2]: splitting bidirectional signals into two unidirectional signals which flow through transistors in opposite directions, and which are then combined into a *winning* signal at each node.

Hayes has developed a theory of distributed switch-level simulation based on *lattice* theory [2], which is capable of handling some of the above mentioned phenomena associated with MOS VLSI. However, the *global* properties of the iteration scheme used to actually derive the steady-state response of the circuit have not been analyzed by Hayes. Indeed, the algorithm converges to the *wrong* state in some fairly straightforward cases as pointed out by Bryant in [1]. In another distributed algorithm, recently proposed [3], the problem of uncontrolled iteration is solved by dealing locally with events in terms of whether they *decrease* or *increase* the path resistance from the source. In the latter case, *pseudo-events* are generated to allow new paths to become dominant. This method is more practical than Hayes' approach and yields the correct results where Hayes' model fails. However, the concept of *pseudo-event* propagation and the informal statement of the algorithm make it very difficult to analyze the limitations and the complexity of the algorithm.

In this paper, we develop a new approach to distributed switch-level simulation based on the concept of *dynamic* graph algorithms. Using this approach, we are able to explicitly formulate the global behavior of the simulation algorithm, and

Manuscript received February 5, 1988; revised December 30, 1988 and August 25, 1989. This paper was recommended by Editor M. R. Lightner.

The author was with the Silicon Design Division, Mentor Graphics, Warren, NJ 07060. He is now with the Tudor Corp., NY, NY 10006.

IEEE Log Number 9040970

compare its results with other graph-based switch-level simulators. In the following section, we present the basic idea behind this approach.

## II. DYNAMIC GRAPH ALGORITHMS

A graph  $G(V, E)$  is commonly defined as a connectivity function over a set  $V$  of vertices and a set  $E$  of edges. Most graph algorithms assume explicitly that the sets  $E$  and  $V$  are constant, and proceed to solve a particular problem defined over these sets. However, a different class of algorithms have been developed to solve problems which arise in dynamic systems. These algorithms have been called *on-line* [4] or *incremental* [5] graph algorithms, and their general characteristic is that they consider the implications of dynamically changing the sets of vertices and edges of the graph.

In general, the class of problems solved by dynamic graph algorithms are of the form: given a graph  $G(V, E)$  which has been subjected to some known static graph algorithm, and to which edges may be added and deleted one by one, is it possible to devise an algorithm which will *update* the information stored on edges and vertices—in less time than it would take to reapply the static algorithm over and over again.

The problem of switch-level simulation has been formulated in graph-theoretic terms by Bryant [6] and Byrd *et al.* [7] by partitioning the circuit into *channel graphs*. The channel graph of a node consists of all nodes (vertices) which are reachable by traversing conducting transistors (edges) from source to drain or vice-versa without crossing power or ground nodes. In both of these works, the problem of finding the steady-state response of a circuit of MOS transistors is solved by applying a static graph algorithm to each *channel graph* every time an event occurs at one of its transistors or nodes. We are, therefore, motivated to ask the following question: given a channel graph  $G(V, E)$ , to which edges can be added and from which edges may be deleted one at a time (corresponding to transistors turning on or off), is it possible to devise an algorithm which will *update* the steady state of all vertices affected by the change, in less time than it would take to reapply the static algorithm to the whole graph?

In the following sections we will develop such an algorithm in several stages. First, we evaluate some of the basic assumptions made in the switch-level model used in [6] and [7]. Then, as we present a more accurate model which handles transistors connected in series more realistically, we proceed to formulate the steady-state response at each node in terms of a *shortest path* problem. We then consider some of the well-known solutions to this problem, and their adaptability to a distributed form. Finally, we develop a method of incrementally updating the graph labels whenever the graph structure changes, which corresponds to handling events of transistors turning on and off in the original circuit.

## III. NETWORK MODEL

The switch-level model is an abstraction of the more detailed electrical model used to describe MOS digital circuits in terms of voltages and currents. At the switch-level, the voltage is abstracted into one of three discrete logic levels where  $L$  represents a low voltage,  $H$  represents a high voltage, and  $X$  represents an indeterminate or unknown voltage. The second state variable, which generally represents the *amount* of charge available at each node, is a complex and unresolved issue among researchers. Although there have been attempts to model *cur-*

*rents* directly at the switch-level, most switch-level models prefer to deal with resistors or attenuators as a means of specifying the *strength* of the signal at each node.

The linear model introduced by Terman [8] gives the basic motivation for viewing a digital MOS circuit as a resistive network. However, from a practical point of view, most simulators, including Terman's own RSIM, do not attempt to directly solve the linear system of equations which represent the resistive network. At best, a voltage divider calculation is carried out, and when a network is found not to be reducible to such a form, *ad hoc* solution methods are used.

The computational complexity of the linear network model, as well as some of the problems encountered in circuits which are not easily reducible to series-parallel form [8], have led to the development of the *dominant path* model by Bryant [6] and, in a slightly different form, by Byrd *et al.* [7]. In this model, it is assumed that the steady state of each node is determined by the *path of least resistance* to a signal source, rather than by the combined resistance of all such paths as in Terman's model. In other words, if two signal paths, parallel or disjoint, lead to the same node—the stronger (less resistive) path will prevail.

Both Bryant and Byrd *et al.* further restrict the model by assuming that only the *most* resistive transistor on each path actually determines the path resistance. It is shown that the switch-level model represents the limiting case of a real resistive network if the discrete set of resistance  $\{r_1, r_2, \dots, r_n\}$  used in any circuit satisfies the *order of magnitude* assumption:  $r_1 \ll r_2 \ll \dots \ll r_n$ . The series-parallel combination rules derived from these assumptions are such that the parallel combination of resistances is defined as their *minimum* (*first assumption*) and the series combination of two resistances is defined as their *maximum* (*second assumption*). The set of possible *path* resistances is, therefore, closed under series-parallel combination giving rise to a fast and efficient solution algorithm.

Both the dominance principle (for parallel drivers) and the order of magnitude principle (for series drivers) can produce overly optimistic results. Fig. 2(b) shows a circuit configuration that would incorrectly be resolved to *high* rather than *unknown*, due to the use of the dominant pull-up resistance rather than the parallel combination. Fig. 2(a) demonstrates the difficulty with the order-of-magnitude assumption. The output value will be resolved to a *high* rather than *unknown*, because the resistances of the pull-up path are not combined in series.

Since the dominant path principle is the cornerstone of all path-based graph algorithms for switch-level simulation, it would be difficult to discard. The order-of-magnitude assumption, on the other hand, can be easily generalized to produce a sum of resistances whenever transistors are connected in series, thereby making the algebra of signals more similar to the algebra of distances used in shortest path algorithms.

## IV. DISCRETE SIGNAL MODEL

A discrete signal is defined as an ordered pair  $\langle s, l \rangle$  where  $s$  is the *strength* and  $l$  is the logic level. The set of possible signal strengths, denoted  $\Psi$ , has  $2n + 2$  members, where  $n$  is a positive integer selected to provide the desired degree of accuracy. The set

$$\Psi = (s_0, s_1, \dots, s_n, s_{n+1}, \dots, s_{2n}, s_{2n+1})$$

which is totally ordered in increasing order from left to right

$$s_0 < s_1 < \dots < s_{2n+1}$$

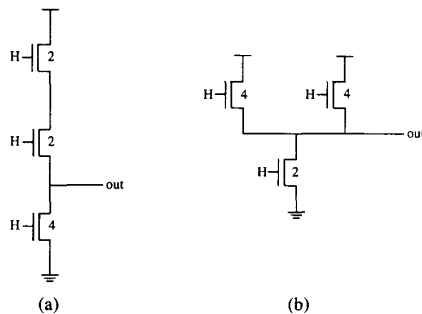


Fig. 2. (a) Attenuation model yields optimistic result of high both yield and optimistic result of high rather than unknown. (b) Attenuation and resistive models.

is partitioned into four groups of signals representing *high-impedance*, *charged*, *driven*, and *supply* strengths, respectively:

$$\Psi = (z, c_1, c_2, \dots, c_n, d_1, d_2, \dots, d_n, S).$$

The high-impedance strength  $z$  denotes zero strength. It is used, for example, to represent the strength driven by an off-transistor. Next are  $n$  capacitive strengths, which are a discretized version of nodal capacitances. Following is a set of  $n$  possible discrete *driven* strengths, and last (strongest), is the *supply* strength attributed to power, ground, and forced input nodes. Generally, the number of charged strengths can differ from the number of driven strengths; however, we will use the same number  $n$  for simplicity.

Two factors directly affect the accuracy of the switch-level model in predicting the correct steady state of a digital MOS circuit: the number  $n$  of different conductance values used, and the algebra of signals defined over the set  $\Psi$ . In Bryant's model [6],  $n$  is restricted to a small integer value, and the algebra  $(\Psi, +, \cdot, z, S)$  is chosen to be a *closed semi-ring algebra*, by interpreting the operators "+" and " $\cdot$ " as the *maximum* and *minimum* operators over the elements of  $\Psi$ .

Since our goal in this paper is to show that switch-level simulation can be formulated as a shortest path problem, we choose to define the *state of a node* as a *vertex label*, and the *resistance* of a transistor as an *edge weight*. Our algebra will be defined accordingly, by choosing " $\cdot$ " as the *minimum* operator, and defining "+" as a true additive operator (defined precisely in the following section).

## V. THE DYNAMICALLY DIRECTED SWITCH MODEL

From a physical point of view, the MOS transistor is never truly *bidirectional*. At any given time, the current  $I_{ds}$  through the channel is determined (approximately) by the voltages  $V_{gs}$  and  $V_{ds}$ , and whenever  $I_{ds}$  is not zero, it flows in a *definite* direction. The phenomenon, which the switch-level model tries to capture, is the MOS transistor's ability to *dynamically* change the direction of current flow through the channel, according to the gate, drain, and source conditions. Based on this argument, we will present a MOS transistor model where the switch is assumed to be *unidirectional* at any given time, but has the ability to dynamically change its direction of signal flow when the occasion calls for it.

For the moment, let us assume that we have determined the *instantaneous* direction of transistor  $t$ , connected between terminals  $x$  and  $y$ , to be  $x \rightarrow y$ . We define the *transfer function* of

the transistor to be a function

$$\delta: \Psi \times \Omega \rightarrow \Psi$$

where  $\Omega$  is defined as a totally ordered set of discrete *resistances*:  $\Omega = (r_1, r_2, \dots, r_n)$  ordered from left to right:  $r_1 < r_2 < \dots < r_n$ . Note that the index  $n$  used here is the same as in the definition of  $\Psi$ . We define the sum of two resistances in the set as

$$r_k + r_l = r_{\min(n, k+l)}.$$

We assume the existence of a modeling function  $r(t): T \rightarrow \Omega$  which assigns to each transistor a discrete *resistance* from the set  $\Omega$ , based on its *width to length* ratio and the process parameters. The function  $\delta$ , which correlates the discrete *strengths* to the set of discrete *resistances*, will be defined, for any strength  $s_k$  in  $\Psi$  as

$$\delta(s_k, r(t)) = s_k - r(t).$$

Since the sets  $\Psi$  and  $\Omega$  are finite, we could explicitly define the subtraction operator used above for any designation of values to the members of the sets  $\Omega$  and  $\Psi$ . However, it is simpler to specify the operation in terms of the set of indexes of  $\Psi$

$$\{0, 1, \dots, n, n+1, n+2, \dots, 2n, 2n+1\}$$

and the set of indexes of  $\Omega$

$$\{0, 1, \dots, n\}.$$

The resistive nature of the transistor is modeled by specifying that for any *driven* or *supply* strength  $s_k$  ( $n+1 \leq k \leq 2n+1$ ), and any transistor resistance  $r_l$  ( $0 \leq l \leq n$ )

$$s_k - r_l = s_{\max(n+1, k-l)}.$$

This describes the fact that driven signals become *weaker* as they pass through a transistor. The problem with defining a subtraction operation over a finite set is that the set may no longer be closed. In practice, we pick the number  $n$  large enough to model chains of up to eight minimum-sized transistors without overflowing, so that mathematically we can assume the set is infinite ( $k-l$  in the equation above will never be less than  $n+1$ ). It will be shown later that the number of different strengths has no effect on the complexity of the simulation algorithm.

Capacitive signals, on the other hand, do not suffer a drop in strength. This is modeled by simply transmitting the incoming signal to the output unaltered. More formally, we can specify that for *every* strength  $s_k$ , where  $k = 0, 1, \dots, 2n+1$ , and for every transistor resistance  $r_l$ , where  $l = 0, 1, \dots, n$

$$\delta(s_k, r_l) = s_k - r_l = \begin{cases} s_{\max(n+1, k-l)} & \text{if } k > n \\ s_k & \text{otherwise.} \end{cases}$$

Fig. 3 shows two examples of strength calculations for a resistive source and a capacitive source. In the first case, the source strength is  $d_7$ , which is a *driven* strength, so the resistive edges cause subsequent vertices to suffer a drop in strength. In the second case, the source is a capacitive node, so no strength degradation occurs.

## VI. DYNAMIC DIRECTION ASSIGNMENT

The wide-spread use of unidirectional switch models and static direction finding programs [9] implies that the number of truly bidirectional transistors in circuits is quite small (typically

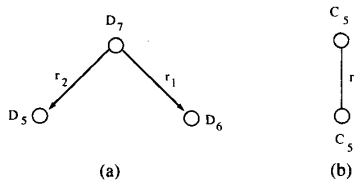


Fig. 3. (a) Strength reduction for driven paths. (b) Strength propagation for charge-sharing paths.

less than 5%), and that undesirable bidirectional effects (such as *sneak* paths) are relatively rare. Simulation speed can be greatly increased by taking advantage of this premise.

The direction of signal flow through a transistor in our model is determined *dynamically* by comparing the signal strengths of the source and drain terminals. By retaining this information internally, the transistor model is able to discern when the direction of signal flow *has changed*, and take appropriate action only if this actually occurs. The model is, therefore, called the *dynamically directed switch* (DDS) model. We define the *instantaneous* direction of an edge in the channel graph as a function of the strengths (vertex labels) of its two endpoints. For two vertices  $u, v \in V$ , with strengths  $s(u)$  and  $s(v)$ , respectively, the direction of the edge  $e_{uv}$  connecting them is defined as

$$\text{direction}(e_{uv}) = \begin{cases} u \rightarrow v, & \text{if } s(u) > s(v) \\ v \rightarrow u, & \text{if } s(v) > s(u) \\ 0, & \text{if } s(u) = s(v). \end{cases}$$

When *both* nodes have the same strength, no direction is assigned, meaning that the edge cannot contribute to the *strength* calculation at either node. If all signal *sources* in the graph were of the same logic level we could *disregard* such edges completely. However, if conflicting sources are present, undirected edges must transmit their logic levels in both directions. Fig. 4 shows an example of an undirected graph, and the directions assigned as a result of applying the above function.

Since the assignment of edge directions is based only on the relative strengths of adjacent nodes, none of the problems associated with static direction finding, such as dealing with barrel shifters, are encountered using this approach.

A *directed path* of length  $k$  is defined as a set of edges  $\langle e_{0,1}, e_{1,2}, \dots, e_{k-1,k} \rangle$ , such that for  $i = 0, 1, \dots, k$  edge  $e_{i-1,i}$  is an edge from vertex  $v_{i-1}$  to vertex  $v_i$  which satisfies: 1)  $e_{i-1,i}$  is directed  $v_{i-1} \rightarrow v_i$ . A *dominant path* is a path which satisfies: 2)  $\delta(s(v_{i-1}), r(e_{i-1,i})) = s(v_i)$  where  $s(v_i)$  is the strength of vertex  $i$ .

Note that, in general, not every directed path is dominant. An edge which is incident on vertex  $v$  and is on a dominant path to  $v$  is said to *dominate*  $v$ . A node may have more than one dominator, but all dominators have the same strength. For example, in Fig. 5, there are two directed paths of length 2 but only the right one is dominant. The left path is *blocked* by the dominant path.

The strength of a path is obtained by combining the strengths of the transistors on the path using their  $\delta$  functions. In the example above, the strength of the dominant path is found by combining the following two equations:

$$d_3 = \delta(d_5, r_2)$$

$$d_2 = \delta(d_3, r_1)$$

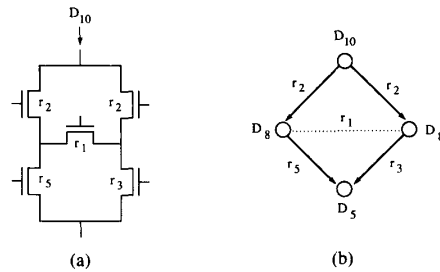


Fig. 4. (a) MOS circuit. (b) Channel graph with strengths and directions assigned, assuming a strength of  $D_{10}$  is being driven into the top node.

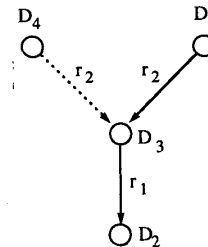


Fig. 5. Directed paths (nondominant path dashed).

which yields

$$d_2 = \delta(\delta(d_5, r_2), r_1)$$

which is equivalent to

$$d_2 = \delta(d_5, r_2 + r_1)$$

where  $d$  stands for a driven strength whose magnitude is given by its index. This property allows us to talk about a path *resistance* as the sum of resistances on the path. We now state this property more generally.

**Lemma 1:** Let  $\langle e_{0,1}, e_{1,2}, \dots, e_{k-1,k} \rangle$  be a dominant directed path of length  $k$  (as defined above). This path can be substituted by a single directed edge  $e_{\text{path}}$  with a combined resistance of

$$r_{\text{path}} = \sum_{i=1}^k r(e_{i-1,i})$$

so that the following equality holds

$$s(v_k) = \delta(s(v_0), r_{\text{path}}).$$

*Proof:* By induction on  $k$ . □

The following lemma shows the practical advantage gained by using the DDS model: we are able to restrict our attention to *directed acyclic graphs*.

**Lemma 2.1:** All directed paths (not necessarily dominant) are acyclic.

**Lemma 2.2:** If a dominant path exists from node  $i$  to node  $j$  (not necessarily directed), then an acyclic dominant path exists from  $i$  to  $j$  with greater or equal strength.

*Proof:* Lemma 2.1 follows directly from the definition of a directed path and the strong monotonicity of the  $\delta$  function over directed edges. Lemma 2.2 follows from the fact that strengths are nondecreasing and positive even for undirected edges. □

A similar result was obtained by Rao and Trick [10] and used to order transistors for execution, such that a transistor will be evaluated only after all its predecessors have been determined. The main difference is that the algorithms in [10] are aimed at statically *partitioning* the network prior to simulation, and therefore, direction and order assignments remain fixed.

## VII. THE SHORTEST PATH PROBLEM

Using the results of the previous section, we can now formulate the problem of finding the *strengths* of all nodes in a channel graph driven by a *single* source as equivalent to finding the shortest path from the source to each node, where "shortest path" is taken to mean "dominant path."

The problem is generally solved by *Dijkstra's* algorithm in  $O(|V|^2)$  time [11] which can be improved to  $O(|V|\log|V|)$  by using Fibonacci heaps [12]. Even faster methods have been devised by Wagner [13] and Johnson [14] for edge-sparse graphs with a restricted set of positive edge weights.

Wagner's FLOW algorithm [13], which forms the basis of Bryant's MossimII algorithm, uses a set of indexed "buckets" to keep marked nodes ordered according to their distance from the source. Then, by always selecting the strongest, it is guaranteed that each edge will only be tested once. Wagner also shows that his algorithm is faster than Dijkstra's only for graphs which satisfy  $|V| \gg n$ , where  $n$  is the number of discrete resistances (edge weights), which is not the typical case in MOS channel graphs.

Dijkstra's algorithm and its variants are not practical candidates for a *distributed* algorithm due to their dependence on a selection process from a global data structure. This class of algorithms uses the *label setting* technique, where a label is never "corrected" after it is set. An alternative class of algorithms, called *label updating* algorithms, use a process of iteratively improving the labels until no further improvement is possible. This class of algorithms is well suited for distributed processing, and has been proved to always terminate for graphs with no negative cycles.

As the basis for our approach, we choose a distributed version of Ford's algorithm [11], which is attributed to Moore [15]. We use Hayes notation  $\#(\text{node})$  to denote the operation of finding the steady state of a node as a function of all signals coming into that node through the channels of conducting transistors.

### Algorithm SHORT

```
(0) short( $u$ ) {
(1)   foreach edge  $e_{uv}$  {
(2)     if ( $s(u) \geq s(v)$ ) {
(3)       if ( $s(u) > s(v)$ ) direction( $e_{uv}$ ) =  $u \rightarrow v$ ;
(4)        $s_{out} = \delta(s(u), r(e_{uv}))$ ;
(5)       if changed  $s_{out}$  que( $v, s_{out}$ );
(6)     }
(7)   }
(8) }
```

The *que* operation will result in evaluating  $s_{out}$  against the other drivers on that node. In most cases this amounts to a single comparison against the node's previous state which is analogous to the fixed-point iteration operation in shortest path algorithms:

$$s_i = \text{minimum}(s_i, s_j + d_{ij}).$$

When the algorithm is executed as a result of a driven turning off, there may be a need to actually perform the  $\#(i)$  operation

when a node event comes off the queue, depending on whether or not that driver was previously a dominator of the node. This case is discussed in more detail in the section on incremental edge deletion.

The worst-case behavior of this algorithm is  $O(|E| \cdot |V|)$ . However, the worst case only occurs in graphs where some nodes have a high *fan-in* factor. In the other extreme, when the graph is a *tree* rooted at *src*, the algorithm is *linear*, and each node only gets updated once. Note that most MOS channel graphs tend to be closer to a tree structure than a clique. Furthermore, once the algorithm is applied to a graph it induces a *shortest path spanning tree*, which makes further updating of the graph labels more efficient.

## VIII. A DISTRIBUTED VIEW

The SHORT algorithm can be also formulated in object-oriented terms as a message passing scheme between two classes of objects: switches (graph edges) and nodes (graph vertices), where switches can only communicate with adjacent nodes and vice-versa.

With this in mind, an MOS network is analogous to a distributed message-passing communication network which uses a shortest path criterion for routing messages from node to node. ARPANET is an example of such a network [16].

The problem most frequently encountered in distributed networks of this type is how to recover from topology changes in the network in an efficient way without using a centralized control mechanism. This problem has been studied extensively and a *linear time* algorithm has been found by Jaffe and Moss [17]. This algorithm, which is essentially a shortest path updating algorithm, can be adapted to the case of MOS circuits using the DDS model presented here.

The challenge of devising a distributed algorithm to update the signals at each node in response to changes in circuit topology (resulting from transistors turning on or off), is in maintaining enough information locally so that objects can make global decisions based on local data changes. The dynamic direction of switches is an example of global information which is retained locally. Other necessary information will be discussed in the next sections.

In addition to finding the dominating strength at each node, the distributed algorithm must also determine the logic levels. Thus the propagation of strengths through switches must be accompanied by a comparison of the logic levels at the drain and source. All *VDD* and *GND* sources can be combined into two sources of equal strength and opposite levels. The problem of strength propagation is then reduced into a single-source problem, and the effect of the logic level on the propagation of signals is seen only when contention occurs. This is handled by comparing the logic levels of signals across directed *and* undirected edges in the graph, creating unknown signals where strengths are equal and levels are different. When unknown signals propagate through the network, they may eventually reach the gate of a transistor introducing a new situation, where the resistance of an edge is indeterminate.

## IX. UNKNOWN SIGNALS

The correct handling of unknown resistances in switch-level networks, arising from transistors with an unknown level at their gate, has proven to be one of the most time consuming aspects of all algorithms. Distributed algorithms have generally avoided the problem by taking an overly pessimistic approach: when-

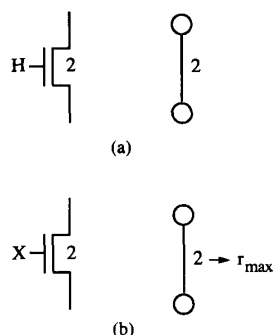


Fig. 6. (a) Fully turned-on switch. (b) Switch with unknown gate.

ever the gate of a transistor is unknown, its output is set to unknown, regardless of the input state. This approach is known to generate "false alarms" for many practical circuits such as NOR gates and PLA's.

Following Terman [8], we model an unknown at the gate of a switch as producing an *interval* of possible resistances, ranging from the maximum resistance (switch is off) to the on-resistance of the switch as shown in Fig. 6. However, since we prefer to remain in the discrete domain, we base our model of unknown signals on the separation of signals into *definite* and *indefinite* signals as in Bryant's theory [6].

A definite signal represents a path of known resistance. This signal can *block* definite or indefinite signals whose strengths are less or equal to its own. An indefinite signal, on the other hand, does not have the ability to block definite signals. It represents a range of strengths from zero up to its value.

The concept of *signal strength*, defined earlier as a single value in an indexed set, is now generalized to a three-tuple as follows:

$$S = \langle s_{def}, s_0, s_1 \rangle$$

where  $s_{def}$  is the definite strength used earlier,  $s_0$  is the strongest definite or indefinite 0-strength, and  $s_1$  is the strongest definite or indefinite 1-strength.

Due to the indefinite components of the generalized strength, we can now only define a *partial order* between strengths where a total order existed before. The partial order is defined as follows:

$$(s_{def}^1 > s_{def}^2) \ \& \ (s_0^1 > s_0^2) \ \& \ (s_1^1 > s_1^2) \ \rightarrow \ S^1 > S^2.$$

If any of these conditions fails to hold, we cannot determine the direction of flow through the transistor and conclude that it is an undirected edge, i.e., the states of the two nodes may affect each other in either direction. The transfer function of the switch is the same as defined for a single strength, except that it is now applied to each of the three signals independently:

$$\delta(S, r) = \langle \delta(s_{def}, r), \delta(s_0, r), \delta(s_1, r) \rangle.$$

The logic level of a node can be determined from the state as follows:

$$\text{level} = \begin{cases} L, & \text{if } s_0 > 0 \text{ and } s_1 < s_{def} \\ H, & \text{if } s_1 > 0 \text{ and } s_0 < s_{def} \\ X, & \text{otherwise.} \end{cases}$$

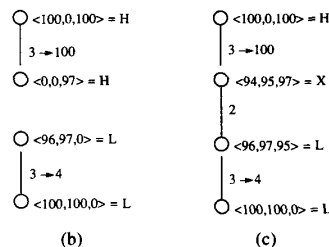
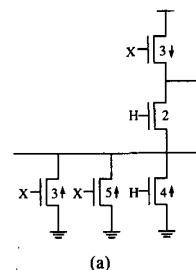


Fig. 7. (a) Sample circuit with switch resistances marked. (b) Signal propagation through switches with marked directions. (c) Signal propagation through middle switch.

Finally, indefinite strengths which are weaker than the definite strength of the node are blocked [6], since there is no need to propagate them any further.

For example, Fig. 7(a) shows a typical circuit (using all NMOS transistors for clarity) with a mixture of known and unknown signals. In Fig. 6(b) the graph is shown prior to evaluating the middle transistor. In this example the maximum strength is 100, which is the strength of  $V_{DD}$  and  $G_{ND}$ . The three pull-downs are represented by a directed edge whose resistance is between 3 and 4. The signal produced by these transistors has a definite strength of 96 ( $100-4$ ), and an indefinite low strength of 97 ( $100-3$ ). The signal  $\langle 96, 97, 0 \rangle$  represents a low signal whose strength ranges from 96 to 97.

Fig. 7(c) shows how the two signals interact when the middle transistor is evaluated.  $s_1$  (95) propagates downward, and  $s_0$  and  $s_{def}$  propagate upward (through a known resistance of 2) resulting in the signals shown. The path-blocking function is then applied to the signal  $\langle 96, 97, 95 \rangle$  which eliminates the weaker  $s_1$  component leaving the signal  $\langle 96, 97, 0 \rangle$ . In practice, a minimum resolution is used when comparing two strengths, such that if two strengths are very close (as in this case) they will actually be considered equal. In the example above, if the resolution is set to more than 1, then both nodes will become unknown—yielding a more conservative result.

## X. DYNAMIC BEHAVIOR

In this section we define the actual algorithms performed by the nodes and switches to determine the global behavior of the circuit. We have shown how switches determine their direction and how they transmit their states to adjacent nodes. We now specify how the switch and node actions are combined to evaluate the state of the whole network.

As we will show in this section, the additional information (besides the signals transmitted by all adjacent switches) which a node must keep track of in order to correctly determine its state, is the identity of the dominant drivers of the node.

### 10.1. Incrementally Adding an Edge

Let  $G_1(V_1, E_1)$  and  $G_2(V_2, E_2)$  be two graphs which have been directed and labeled by the SHORT algorithm. And let us assume that there are no unknown signals involved. The graphs are either mutually exclusive or  $G_1 = G_2$ . Each vertex is labeled by  $s(v)$  and each edge is directed and labeled with its resistance. The graphs, as we have claimed, possess no directed cycles.

Now assume that a transistor which was formerly off—turns on. This corresponds to the creation of a new edge  $e_{vu}$ . Let  $v \in V_1$  and  $u \in V_2$ . The addition of the edge  $e_{vu}$  will result in a new graph  $G(V, E)$ , whose vertices are  $V = V_1 \cup V_2$ , and whose edge set is  $E = E_1 \cup E_2 \cup \{e_{vu}\}$ . The situation is depicted in Fig. 8. Since both graphs were assumed to have reached their steady state, the direction of the new edge  $e_{vu}$  is found by comparing  $s(v)$  and  $s(u)$  as defined earlier. Assume that the edge is found to be directed  $v \rightarrow u$ .

When the edge  $e_{vu}$  turns out to be a new single dominator of  $u$ , only the subgraph rooted at  $u$  node may have to be updated. This is accomplished by *queuing* the target node for execution of Algorithm SHORT, and allowing the shortest path calculation to proceed from there. The important point to note here is that once the direction of the edge  $e_{vu}$  is determined to be  $v \rightarrow u$ , the state of node  $v$  and all its predecessors cannot be changed by adding the new edge because there can be no directed feedback paths from  $G_2(V_2, E_2)$  to  $G_1(V_1, E_1)$  which affect the dominant path to  $v$  (this would result in a directed cycle). If the edge  $e_{vu}$  is found to be undirected ( $s(v) = s(u)$ ), then the state of *both* nodes may change, causing both nodes to reconsider their state. This time the effects of the changes will only propagate through undirected edges or edges directed outward. The algorithm for edge addition can be summarized as follows.

- 1) Find direction of newly added edge. Calculate its output signal.
- 2) If this signal is a new *single* dominator of the output node, propagate new signal downstream.
- 3) Otherwise check and propagate level contention.

Fig. 9 shows an example of incremental edge addition. When the top transistor turns on, its direction is found to be left-to-right by comparing  $s_1 = d_8$  with  $s_2 = d_6$ . Since the newly added edge has become a single dominator of the output node, all nodes further down the path must be updated.

If the resistance of the switching transistor were  $r = 1$  instead of 2, it would become a new single dominator of  $n_2$ , and all node strengths further down the path would be re-evaluated by applying the SHORT algorithm from  $n_2$ .

### 10.2. Incrementally Deleting an Edge

Incremental edge deletion due to a transistor turning off is slightly more complicated because it may cause one signal path to be broken, and another path to over-ride it at some other node. Furthermore, when a path is broken, all signals which are further down the path are no longer valid, and even their edge directions may no longer be valid. On the other hand, if an edge,  $e_{uv}$  (directed  $u \rightarrow v$ ) which is *not* a dominator or  $v$  is deleted, node  $v$  and all nodes further down the path will not be affected at all.

The problem then becomes, how to consistently keep track of the dominators of each node. This is solved by keeping, for each transistor output terminal, a marker indicating whether or not it is *dominating* the output node, and a dominator count for each node. When a single dominator is removed from a node,

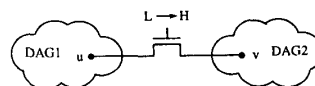


Fig. 8. Incremental edge addition: if  $s(u) > s(v)$ , update only DAG2. If  $s(v) > s(u)$ , update only DAG1, otherwise propagate logic levels—if different.

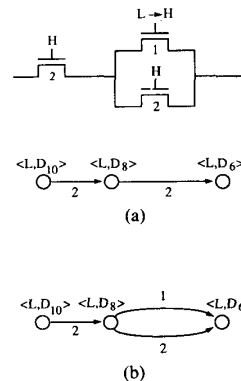


Fig. 9. (a) Channel graph before transistor turns on. (b) Incremental effect of transistor turning on.

its previous signal is invalidated and the node must find a new dominator. If none exists, the node will remain at its old logic level, with a charged strength proportional to its capacitance. All this is built into the  $\#(\text{node})$  operation mentioned earlier. The algorithm can be summarized as follows.

- 1) If the deleted edge was not a *single* dominator of its output node, update dominator count of the node (if required).
- 2) Otherwise find new dominators, queue the dominator signal for propagation to all adjacent edges.

Fig. 10 will be used to demonstrate the edge deletion algorithm. In Fig. 10(a), the steady state is shown before the transistor turns off: the output node is driven to  $\langle L, d_6 \rangle$  by two dominators. When the transistor turns off, it is found to have been a single dominator of its output, so the state of that node becomes charged since no other transistors are directed into the node. Both outgoing transistors are then scheduled. Assume the top transistor runs first, it determines that it *was* a dominator, of its output but not a *single dominator*, so the dominator count of the output is decremented. Now the other transistor runs, and this time it *is* a single dominator of its output, so the signal  $\langle L, d_6 \rangle$  is invalidated and  $n_2$  settles to its charged state. If the charge of the output is greater than  $c_4$ , it would become a new dominator, and propagate backwards through the two transistors.

When deleting a single driver from a graph that may potentially have cycles, care must be taken to guarantee that all signals which are no longer valid be invalidated prior to propagating their states, otherwise they may no longer represent actual dominant paths and Lemmas 2.1 and 2.2 may not hold (i.e., cycles may be created). For example, in Fig. 10(b) the "backward" propagation of signal  $\langle L, d_6 \rangle$  must be avoided. This problem is solved by a simple rule: any node evaluation which results in a charged state is propagated immediately, whereas driven states are deferred by queuing them on a zero-delay queue. Thus all traces of the previous dominant path are deleted before the secondary drivers propagate their new dominant strengths. In Fig. 10(b) this means that the evaluation of

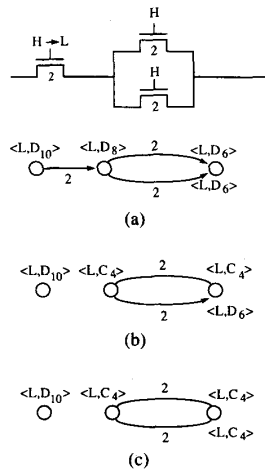


Fig. 10. (a) Channel graph before transistor turns off. (b) Edge deleted, path through top transistor evaluated. (c) Path through bottom transistor evaluated, single dominator was deleted—circuit settles to charged state.

the output to  $\langle L, d_6 \rangle$  will not propagate “backward” until the charged state propagates through the other switch as well, at which time the driven event would be canceled by zero-delay spike filtering. This problem does not exist, of course, when RC delays are used.

#### XI. CORRECTNESS AND COMPLEXITY

Shortest path updating algorithms like the one described here have been studied extensively in connection with routing problems in distributed packet switching network. A proof of correctness of a similar updating procedure is given by Tajibnapis [16] for the case of equal edge costs. It is also shown there that the algorithm remains correct even if edge events occur while the effect of other events are still being updated, and that the algorithm behaves correctly on startup. Jaffe and Moss [17] have shown that the updating procedure for communication networks takes *linear* time if topological order is imposed during the update procedure.

In an event-driven simulation environment it is impossible to prove that the iterative process will always terminate due to the possibility of simulating circuits with zero-delay feedback paths which cause oscillations. However, the algorithms proposed here offer extra degrees of control over the iteration process, by keeping track of changes in signal-flow direction through transistors, and dominators of each node. This imposes an order on the updating process making it more efficient, on the average, than reapplying the static algorithm. It is difficult to derive the complexity of this algorithm since it depends strongly on the circuit graph. For example, if an edge is added near a leaf of a long path, few updates will be necessary, whereas if an edge is added near the root more updates will occur.

For the case of edge addition where a path to VDD or GND exists, since each of the subgraphs is a DAG the updating proceeds in topological order and takes linear time. When edges are deleted, the algorithm first traces concurrently in all directions until it finds a new dominator, and then proceeds updating from there. The analogous action in the mossim II algorithm is a depth-first search to find the bounds of the channel graph. Thus in the worst case the updating procedure will have to do the same amount of work as the static algorithm, whereas in

most cases it will terminate the search earlier—when a dominator is found.

#### XII. TIMING SIMULATION

The need to model the time-domain behavior of large digital circuits has given rise to switch level timing simulation methods. These methods offer very limited accuracy compared to analog circuit simulation; however, it becomes possible to estimate the overall performance of large circuits whose size prevents the application of analog simulation techniques.

First-order timing models are available for *tree* networks [18] as well as for some more general circuit topologies [19]. Higher order models [20] are also available, but their cost is usually higher than the accuracy gain. Most switch level *simulators*, which do include a timing model, use the RC tree approximation with or without side-branch capacitances [21]. Apart from the problems of the model itself, the application of a timing model within an event-driven simulation environment is only partly justifiable due to the fact that simulator is only guaranteed to find the *steady-state* response of the circuit under certain assumptions. One of these assumptions is that the *gates* of transistors are held fixed while signals are propagating through the transistor drain-source path. This does not hold true in timing simulation, and may lead to false resolution of races. Another problem which arises in timing simulation is *dynamic* charge sharing effects and, generally, delay assignment to charge sharing events. Both of these problems have no simple consistent solution, and are avoided altogether by not considering dynamic charge sharing at all, and assigning zero delay to all static charge sharing events.

In spite of these problems, reasonable results for large classes of circuits can be achieved with very little computational overhead. The path resistance is embedded within the propagated signal, and the *que* function can be interpreted as placing nodes on a time queue as explained previously. Since the algorithm's complexity is independent of the number of discrete resistances used, a large set of resistances may be used to obtain a high level of contention resolution, and realistic delays. The problem of false intermediate states then becomes mainly a problem of correctly handling scheduling, unscheduling, and rescheduling of nodes in transition.

#### XIII. THE LSIM MIXED-MODE SIMULATOR

The switch-level algorithm described in this paper has been implemented within the *Lsim mixed-mode analog and digital simulator* [22]. The *Lsim* simulator is an extensible simulation environment which allows many different simulation algorithms and models to co-exist and exchange information at their least common denominator level.

For digital simulation *Lsim* supports all levels from high-level behavioral modeling, through several levels of register-transfer and gate-level logic modeling, down to the switch-level simulation model described here. On the analog side, the ADEPT timing simulation algorithm is used for fast and accurate circuit simulation of digital circuits [23], and a variety of SPICE-level direct solution algorithms have been integrated for true analog circuits. All simulation levels, except for the analog algorithms, utilize the same basic philosophy of distributed models communicating through a common node-arbitration mechanism. The analog algorithms are more independent in that they operate on selected portions of the circuit, communicating with the rest of the circuit via the *simulation manager* interface, which is described in [24].

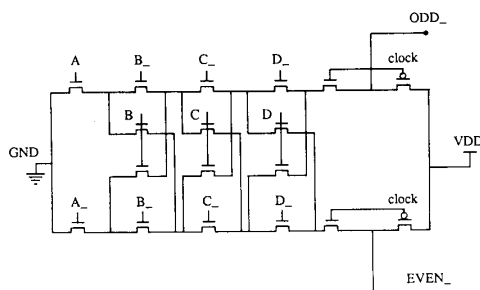


Fig. 11. Parity generator circuit example.

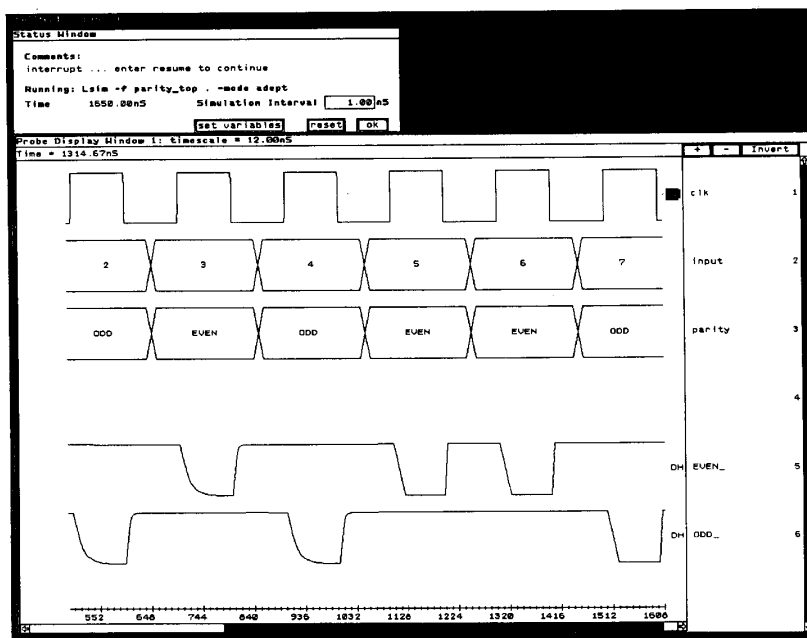


Fig. 12. Simulation of parity generator circuit. The circuit is dynamically toggled from ADEPT timing simulation mode to switch-level RC mode.

Within an environment such as this, the main task of switch-level simulation is not to provide accurate timing and voltage waveforms, but rather to provide fast functional verification of transistor level designs embedded within a hierarchical environment which may also include higher level behavioral models. Parts of the circuit which cannot be adequately modeled at the switch level such as bootstrap drivers, sense amplifiers, etc., can either be substituted by a behavioral description or designated for ADEPT simulation, which is fast enough to make such a scheme practical. Switch-level simulation is also used to initialize large transistor circuits for ADEPT timing simulation. The ADEPT algorithm is also activity-based, and therefore, slower at startup than at any later time where the activity level is much lower.

#### XIV. RESULTS AND APPLICATIONS

The simulation algorithm described in this paper, as implemented in the Lsim simulator, has been tested on a large variety of circuits and numerous full custom chips. Some of these include Motorola's MC88100 32-b RISC CPU chip with over 165 000 transistors, and the MC88200 Cache Memory Man-

TABLE I  
LSIM DDS SIMULATION BENCHMARKS

Circuit	Trans	Cycles	CPU (minutes)	Events/Seconds
static alu	1100	90	0.4	12 166
addr data-path	2200	60	0.5	11 692
i/o data-path	6100	15	0.2	12 220
multiplier	12 000	45	4.2	13 400
mixed chip	16 200	200	5.5	12 240
micro store	48 700	13	1.8	14 100

agement Unit (CMMU) which has 750 000 transistors. In the CMMU chip, the actual memory array (which accounts for most of the transistors) was simulated in Lsim as a behavioral model, with the rest of the chip running in switch mode. The algorithm has proven to be fast, accurate, and helpful in interactive debugging. The instantaneous transistor directions make it very easy to trace signals to their sources when some part of the circuit behaves erroneously.

Table I shows simulation statistics for a number of small to

medium sized circuits. All benchmarks were performed on a SUN-4 with 16 MByte of main memory. The *mixed chip* entry represents an actual chip where only two blocks (including the data path) were simulated in transistor mode with the other blocks simulated as behavioral models.

When compared to global algorithms such as MossimII, it is possible to come up with examples where the algorithm will not provide any improvement over the *memoryless* approach. However, for typical channel graph and a typical mix of transistor events, the total number of label updates is found to be significantly smaller using the algorithms presented here than using the global approach.

As an example of the efficiency of incremental updating in MOS circuits, consider the CMOS parity-generator circuit in Fig. 11, which is made up of dynamic XOR gates with shared terms. The simulation results for this circuit are shown in Fig. 12 using the multimode capabilities of Lsim. First, the circuit is run in the ADEPT mode for several cycles, and then all transistors are toggled into DDS mode. Note the accuracy of the delay calculations in DDS mode in comparison to the ADEPT waveforms. During the evaluation phase, the signal *clk* goes high, causing two n-channel transistors to turn on and two p-channels to turn off. When the n-channel transistors turn on, a path to ground is created either to *ODD\_* or to *EVEN\_*. Updating this path requires only one operation, because the signal is already "waiting" at the input of the enabling transistor.

## XV. CONCLUSIONS

A new distributed switch level simulation algorithm has been developed which is event-driven at the transistor level. Each transistor only uses information available at its three terminals, and each node only communicates with adjacent transistor terminals. Each transistor element is modeled as a DDS, transmitting signals from the stronger terminal to the weaker. The nodes are able to determine their next state by keeping track of all drivers dominating the node.

We have shown that the criteria used to determine the next state is not arbitrary. It is defined in such a way that the whole circuit when viewed as a *graph*, adjusts its edge and vertex labels to reflect connectivity changes which correspond to transistors turning on and off. The simulation algorithm has been formulated in terms of the events which may occur in the graph: edge addition, edge deletion, and changes in signal values.

## ACKNOWLEDGMENT

The author would like to thank Carl Christensen, Hal Alles, Peter Odryna, Kevin Nazareth, Mike Purnell, and Mehmet Cirit for conceiving and creating the Lsim mixed-mode simulation environment.

## REFERENCES

- [1] R. Bryant, "A survey of switch level algorithms," *IEEE Design Test*, Aug. 1987.
- [2] J. Hayes, "Pseudo-Boolean logic circuits," *IEEE Trans. Comput.*, vol. C-35, July 1986.

- [3] R. Sunblad and C. Svensson, "Fully dynamic switch level simulation of CMOS circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, Mar. 1987.
- [4] S. Even and Y. Shiloach, "An on-line edge deletion problem," *J. ACM*, vol. 28, Jan. 1981.
- [5] G. Cheston, "Incremental algorithms in graph theory," Ph.D. dissertation, Dept. Comput. Sci. TR 91, Univ. of Toronto, Canada, Mar. 1976.
- [6] R. Bryant, "A switch level model and simulator for MOS digital systems," *IEEE Trans. Comput.*, vol. C-33, Feb. 1984.
- [7] R. Byrd, G. Hachtel, M. Lightner, and M. Heydemann, "Switch level simulation: Models, theory, and algorithms," in *Advances in Computer-Aided Engineering Design*, A. Sangiovanni-Vincentelli, ed. London, England: JAI, vol. 1, 1985.
- [8] C. Terman, "Simulation tools for digital LSI design," Ph.D. dissertation, VLSI Memo no. 83-154, Mass. Inst. of Tech. Sept. 1983.
- [9] N. Jouppi, "Derivation of signal flow direction in MOS VLSI," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 480-489, May 1987.
- [10] V. Rao, and T. Trick, "Network partitioning and ordering for MOS VLSI circuits," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, Jan. 1987.
- [11] S. Even, *Graph Algorithms*. Computer Science, 1979.
- [12] M. L. Fredman and R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," *J. ACM*, vol. 34, no. 3, pp. 596-615, July 1987.
- [13] R. Wagner, "A shortest path algorithm for edge sparse graphs," *J. ACM*, vol. 23, Jan. 1976.
- [14] D. Johnson, "Efficient algorithms for shortest paths in sparse networks," *J. ACM*, vol. 24, Jan. 1977.
- [15] U. Pape, "Implementation and efficiency of Moore-algorithms for the shortest routes problem," *Math. Program.*, vol. 7, pp. 212-222, 1974.
- [16] W. Tajibnapis, "A correctness proof of a topology information maintenance protocol for a distributed computer network," *Commun. ACM*, vol. 20, pp. 477-485, July 1977.
- [17] J. Jaffe and F. Moss, "A responsive distributed routing algorithm for computer networks," *IEEE Trans. Commun.*, vol. COM-30, pp. 1758-1762, July 1982.
- [18] P. Penfield, Jr. and J. Rubinstein, "Signal delay in RC tree networks," in *Proc. 18th Design Automatic Conf.*, 1981, pp. 613-617.
- [19] T. Lin and C. Mead, "Signal delay in general RC networks," *IEEE Trans. Computer-Aided Design*, vol. CAD-3, pp. 331-349, Oct. 1984.
- [20] M. Horowitz, "Timing Models for MOS Circuits," Ph.D. dissertation, Stanford Univ. Dept. of Comp. Sci., 1983.
- [21] P. Dewilde, A. Van Genderen, and A. DeGraff, "Switch level timing simulation," in *Proc. ICCAD*, 1985.
- [22] P. Odryna, K. Nazareth, and C. Christensen, "A workstation-based mixed mode circuit simulator," in *Proc. 23rd Design Automation Conf.*, 1986.
- [23] P. Odryna and S. Nassif, "The ADEPT timing simulation algorithm," *VLSI Syst. Design*, Mar. 1986.
- [24] J. Griffeth and P. Odryna, "Versatile manager unifies simulators," *High Performance Syst.*, June 1989.



**Dan Adler** received the B.S.E.E. degree in computer engineering from the Technion in Haifa, Israel and the M.S.E.E. degree from Rutgers University.

He was a manager of Simulation Algorithm Development with the Silicon Design Division at Mentor Graphics, Warren, NJ, where he was responsible for the development of logic and switch-level simulation algorithms and timing analysis. Currently, he is with the Tudor Corp., NY, NY. His interests include parallel switch-

level simulation.