We derive a linear-time race detection algorithm for switch-level circuits that have no feedback within a clock phase, and have unit fanout. We show that the problem becomes NP-complete if fanout of two or more is allowed.
A Linear-Time Algorithm for Switch-Level Simulation with Race Detection in a Class of MOS VLSI Circuits

Vijaya Ramachandran

Coordinated Science Laboratory
University of Illinois at Urbana–Champaign
Urbana, IL 61801

ABSTRACT

We derive a linear-time race detection algorithm for switch-level circuits that have no feedback within a clock phase, and have unit fanout. We show that the problem becomes NP-complete if fanout of two or more is allowed.

This research was supported in part by DARPA under ONR N00014-82-K-0549 while the author was at Princeton University. Preparation of this report was supported by NSF under Grant ECS-8404866. A preliminary version of this work was presented at the IEEE International Conference on Computer Design: VLSI in Computers (ICCD '83), Port Chester, NY, October 31–November 3, 1983.
A Linear-Time Algorithm for Switch-Level Simulation with Race Detection in a Class of MOS VLSI Circuits

Vijaya Ramachandran

Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
Urbana, IL 61801

ABSTRACT

We derive a linear-time race detection algorithm for switch-level circuits that have no feedback within a clock phase, and have unit fanout. We show that the problem becomes NP-complete if fanout of two or more is allowed.

1. Introduction

VLSI design verification tools are a crucial component in the production of complex circuits. In this paper, we present a fast and provably correct algorithm for one design verification tool: digital simulation and race detection in a class of MOS VLSI circuits.

Digital simulation of MOS VLSI circuits requires techniques different from those used in logic-gate simulation, since certain structures commonly used in MOS circuits (notably pass transistor logic) are not boolean gates. The MOS circuit is thus, more appropriately modeled as an interconnection of transistors, which is commonly called a switch-level circuit, since a transistor basically behaves as a switch which is on or off, depending on the state of its gate. Similar structures have been studied in the past in the context of contact networks that model electro-mechanical relays. However, there are some

This research was supported in part by DARPA under ONR N00014-82-K-0549 while the author was at Princeton University. A preliminary version of this work was presented at the IEEE International Conference on Computer Design: VLSI in Computers (ICCD '83), Port Chester, NY, October 31 - November 3, 1983.
differences in the two models. Further, the problem of detecting races was not studied in the context of relays, since relays are quite slow, and delays are easy to control; also, contact networks were small, and so race-free networks were easy to build. The situation is quite different in large, high-speed transistor networks, and thus new simulation techniques are required for switch-level circuits.

Switch-level simulation is an essential component of a MOS VLSI design verification system. Typically, this simulation is done at two stages in the design process. After a circuit is designed, but before it is laid out, the design is passed through the switch-level simulator to ensure that its digital behavior is correct. After the circuit is laid out, standard circuit extractors (for example, the Berkeley CAD tool MEXTRA) recover the structure of the circuit from the layout as an interconnection of transistors. This extracted circuit is simulated once again at the switch-level. This second test is essential, since it simulates the structure of the circuit as defined by its layout. Often, inadvertent shorts between wires and other errors are introduced during the layout process, and these need to be detected and corrected. Thus fast, reliable algorithms for switch-level simulation are essential in a MOS VLSI design verification system.

2. Switch-Level Model and the Race Detection Problem

For the purposes of switch-level simulation, we view the transistor as a device with three terminals: source, gate, and drain. A high voltage at the gate (denoted by logical level 1) causes a high conductance path between the source and the drain, and the transistor is said to be on. A low voltage at the gate (denoted by logical level 0) isolates the source from the drain, and the transistor is said to be off. When the gate is at 1, the source and drain settle at a common voltage that is determined by their current values (we ignore any small voltage drop across a transistor that is on, since switch-level simulation models the digital behavior of the circuit). This depends on the logical levels of the two nodes as well as the relative strengths of the two values, and the logical output is usually specified by a merge table.

The switch-level abstraction attempts to capture the digital behavior of the transistor switch. This is achieved here by associating a logical value for each node at any instant of time, which depends on
its strength as well as its logical level.

The strength of a node indicates the extent to which it can supply a voltage level on its own. **Input nodes** are the strongest, and these are nodes for which the signal comes from an external source. In nMOS, a node connected to a voltage source through a pullup transistor is called a *pullup node*. This is a strong node, but not as strong as an input node. This node is at logical level 1 except when it is connected to an input node through a sequence of transistors that are on, in which case the node temporarily takes on the value of the input node. Any other node is a **weak node**. Among weak nodes, some (e.g., buses) have much larger capacitance than other nodes, and hence the level on a bus dominates the level of any other weak node that it is connected to.

Some authors (e.g., [Bry84]) do not have an explicit pullup node, but instead use different types of transistors to model the same effect. The results in our paper can be easily modified to hold under this model, too.

The difference between a strong and a weak node is that a strong node has a unique level of its own when it is isolated from other nodes, while a weak node 'remembers' the last level that it had before it was isolated. This property of weak nodes is a crucial one in causing race conditions, as we shall see later. We will call the unique level of a strong node its **intrinsic level**, and this is specified as part of the structural description of the circuit.

We generalize the above properties in our model. We assume that the technology can support \( r \) different strengths, \( S_r > S_{r-1} > \ldots > S_1 \), where the last \( q \) values are weak, and the first \( p = r - q \) values are strong. We assume that the strengths are sufficient in number and type so that any \( n \)-input boolean gate (AND or NAND, OR or NOR) can be constructed by interconnecting nodes of appropriate strengths with a number of transistors that is linear in \( n \). This is certainly the case if \( r \geq 3 \) and \( p \geq 2 \) (since \( r=3 \) and \( p=2 \) models nMOS).

In boolean circuits, we have two digital levels, 0 and 1, and this is the behavior we wish to model. However, it turns out (as we will see later) that it is necessary to introduce a third intermediate level \( u \) in the switch-level model. Thus, there are three digital levels: 0, 1, and \( u \). An undefined level, \( u \), is a logical level that corresponds to an
intermediate voltage level; it may represent a logical 0, a logical 1, or neither. The merge of two values of the same strength that are different or that have undefined levels results in an undefined level at the same strength. Otherwise, the merge of two values results in the stronger value. Thus, given the set of strengths that model our technology, we can specify the merge table using the above two rules. A table of the merge rules for nMOS (with one weak strength) is given in Table 1. We indicate the strength of a value by a superscript: in this case, \(i\), \(p\), and \(w\) stand for input, pullup, and weak strengths, respectively.

<table>
<thead>
<tr>
<th></th>
<th>(0^i)</th>
<th>(1^i)</th>
<th>(u^i)</th>
<th>(1^p)</th>
<th>(0^w)</th>
<th>(1^w)</th>
<th>(u^w)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(0^i)</td>
<td>(0^i)</td>
<td>(u^i)</td>
<td>(u^i)</td>
<td>(0^i)</td>
<td>(0^i)</td>
<td>(0^i)</td>
<td></td>
</tr>
<tr>
<td>(1^i)</td>
<td>(u^i)</td>
<td>(1^i)</td>
<td>(u^i)</td>
<td>(1^i)</td>
<td>(1^i)</td>
<td>(1^i)</td>
<td></td>
</tr>
<tr>
<td>(u^i)</td>
<td>(u^i)</td>
<td>(u^i)</td>
<td>(u^i)</td>
<td>(u^i)</td>
<td>(u^i)</td>
<td>(u^i)</td>
<td></td>
</tr>
<tr>
<td>(1^p)</td>
<td>(0^i)</td>
<td>(1^i)</td>
<td>(u^i)</td>
<td>(1^p)</td>
<td>(1^p)</td>
<td>(1^p)</td>
<td></td>
</tr>
<tr>
<td>(0^w)</td>
<td>(0^i)</td>
<td>(1^i)</td>
<td>(u^i)</td>
<td>(1^p)</td>
<td>(0^w)</td>
<td>(u^w)</td>
<td>(u^w)</td>
</tr>
<tr>
<td>(1^w)</td>
<td>(0^i)</td>
<td>(1^i)</td>
<td>(u^i)</td>
<td>(1^p)</td>
<td>(u^w)</td>
<td>(1^w)</td>
<td>(u^w)</td>
</tr>
<tr>
<td>(u^w)</td>
<td>(0^i)</td>
<td>(1^i)</td>
<td>(u^i)</td>
<td>(1^p)</td>
<td>(u^w)</td>
<td>(u^w)</td>
<td>(u^w)</td>
</tr>
</tbody>
</table>

Switch-level simulation ignores the low-level electrical aspects of circuit simulation and deals with the logical aspect only. An early switch-level simulator for MOS circuits is the well-known one introduced by Bryant [Bry80]. Our model is similar to this, and we describe it below. We model the structure of the circuit as a set of nodes connected by transistors. We partition this network into a set of transistor groups. Nodes within a transistor group are interconnected through the source-drains of transistors. Between transistor groups, only unilateral interactions occur from a node in one group to the source and drain of a transistor in another group of which this node is the gate.
We thus have two graphs associated with the structure of the circuit. The first is an undirected multigraph whose nodes are the nodes of the circuit, and in which an undirected edge is introduced to connect the source and drain of each transistor. The connected components of this graph are the transistor groups. Since we have a one-to-one correspondence between the set of transistors and the set of edges in this graph, we will apply the terminology for transistors to these edges, e.g., we will talk of edges being 'on' or 'off', referring to the transistors they represent.

The second graph associated with the circuit is a directed multigraph whose nodes are the transistor groups, and in which there is a directed arc from transistor group i to transistor group j exactly when there is a transistor whose gate is in transistor group i and whose source is in group j. We call this graph the control graph of the circuit.

Each phase of a clock cycle is simulated by setting the input and clock nodes to their new values, and then finding the common value for the connected components in a transistor group in relation to the on transistors (using the merge rules), each time the gate of some transistor in the group changes level. It is clear that a single transistor group may be resimulated many times during a clock phase, if the gates of its transistors change state at different times during the simulation. Thus it is not possible to put an a priori bound on the running time of this simulator.

The control graph of the circuit will be acyclic (i.e., a DAG) if there is no feedback in the circuit. In most well-designed circuits, there is no feedback during a clock phase, since the clocks are usually placed so that the clock that is off breaks all feedback paths. Under this assumption, [LiSeVa81] devised a faster algorithm for simulation. This algorithm finds an ordering for the transistor groups under each clock phase, so that if the gate of a transistor lies in transistor group i and its source and drain lie in transistor group j, then i<j. With this ordering, the simulation of each clock phase can be done in one pass through the network. [LiSeVa81] carry out the simulation by implementing the merge rules by a fast union-find algorithm. The correctness of the simulation follows from the fact that, within a connected component, the merge of the source-drains of transistors that are on can be done in any order, and the final values of the nodes will
remain the same, since, when we simulate the merge rules on the ith transistor group, all the gates of transistors whose source-drains are in this ith component have already been updated. The simulator looks at those transistors in this group whose gates are at logical level 1 and merges the corresponding source-drains. It is easy to see that the final value will remain the same, regardless of the order in which the nodes are merged. We state the result in the following:

**Proposition 1:** If, in a transistor group, the values of the source-drains of on transistors are merged according to the merge rules of Table 1 or its generalization, then the final value at any node in the group is independent of the sequence in which the merges are implemented.

**Proof** The result follows immediately from the fact that the merge operator of Table 1 is associative and commutative.

Since, after a merge, both nodes get the same value, they lie in a common class. This suggests the use of a set-union algorithm to implement the merge, where two nodes will eventually lie in the same set if and only if they are connected through a sequence of transistors that are on. A well-known algorithm for this runs in time \(O(TG(T))\) per clock phase, where \(T\) is the number of transistors in the circuit, and \(G\) is the inverse Ackermann function, which is a function that grows very slowly \([\text{AhHoU174}]\). This is the algorithm used by \([\text{LiSeVa81}]\) for the simulation, and we will call this the LSV simulator. We will refer to any circuit that has no feedback loop within a clock phase as a nonoscillating circuit, since such a circuit will settle to a steady state in each clock phase. It turns out that the simulation can actually be done in linear time since we do not require the full power of the union-find algorithm. It is sufficient to carry out a DFS (Depth First Search) on the transistor group to find the connected components while retaining only edges for transistors that are on, and keep information on the common value for each connected component.

One problem with the LSV simulator (as also with the simulator in \([\text{Bry80}]\)) is that it does not detect race conditions. A race condition occurs at a node in a circuit if differing delays along different paths to the node cause the node to settle at different values. Basically, this means that a race condition occurs at a node if its final value depends on the sequence of switching of the gates in the circuit. In
logic gate circuits, there can be no race condition in the absence of feedback. However, this does not hold for transistor switch-level circuits, because of charge sharing. An example of a race condition in a circuit is given in figure 1. Given Proposition 1, it may be a bit surprising that a race condition can occur in a nonoscillating circuit. However, though Proposition 1 guarantees the uniqueness of the value at a node, regardless of the sequence in which the on gates switched, it assumes that gates that switched off did so before the gates that switched on, and does not look into the sequence of switchings of the gates at level 0. Thus it does not take into account the fact that a gate at level 0 may have switched off only after some other gates switched, and this may give rise to a non-unique value at a node. In fact, this is exactly the source of the problem in figure 1. Here, nodes a and b are initially at level 1, and this gives rise to level 0 at output c. In the next clock phase, both a and b switch to 0. If a switches before b, then the value at c remains 0. But if b switches before a, then c is connected briefly to a pullup node that is not grounded, and so goes to logic level 1, and retains this value after a switches off.

An example of a race condition in nMOS

The problem of detecting races in MOS VLSI circuits is particularly important since delays are introduced in many ways: through transistor
switching times, through delays along long wires in the layout, through parasitic capacitances, etc. Many of the causes for delay depend not only on the structure of the circuit, but also on the layout chosen for it. It is thus highly desirable to construct a circuit that will perform correctly independent of individual delays.

We now introduce the concept of an unknown level. We said earlier that an undefined value occurs when there is a merge of a 0 and a 1 of the same strength. In the presence of a race condition, we have another problem. The correct level at a node cannot be determined if two different paths to the node give rise to say, 0 and 1 respectively, at that node. However, in any real operation of this circuit, the value at the node can be either a 0 or a 1, i.e., the node need not be at an intermediate voltage level. It is simply that we cannot tell by simulation what that level will be, and indeed, that level may vary at different times on the same circuit. We shall call this an unknown level, and denote it by the letter w. An unknown level represents a set of levels, and w can equal one of the following sets: {0,1}, {0,u}, {1,u}, or {0,1,u}.

Notice that, for any gate with an undefined or unknown level, the corresponding transistor can be either on or off. This is the critical property we will use in our race detection algorithm. So, for the rest of this paper, we will use the symbol x to denote the presence of u or w (or both). We will also not attempt to distinguish between u and w. Hence our algorithm does not explicitly inform us whether x represents u or w (or both), but it does correctly detect all occurrences of u and w. The algorithm can be extended to correctly identify levels u and w, but we have not included those details in this paper.

An important feature in the production of races is the presence of hazards in the circuit. A hazard occurs at a node during a clock phase if the node can switch through more than one level before it settles to its final steady-state value. It is well-known [Mow76] that the presence of a hazard in a purely combinational boolean circuit does not affect its steady-state performance. However, because of the presence of charge-sharing in transistor switch-level circuits, the above result does not carry over to the switch-level case. Figure 2 gives an example of a situation when a hazard at a node results in a race condition at another node, even though there is no feedback in the circuit. The figure shows the same circuit as figure 1, but here a is initially at 1 and
b at 0. In the next clock phase, a switches to a 0 while b remains at 0. It would appear that c will have to remain at 1. However, if there is a hazard at b, the value at b will go to level 1 for a short time before returning to 0. If b makes the transition to 1 after a switches to 0, then c is connected to an input node at level 0 for a short time, and so goes to level 0, which it retains after b switches off; otherwise if b completes the transition to 1 before a switches to 0, then c remains at level 1. Thus we have a race condition at c caused by a hazard at b.

![figure 2](image)

An example of a race condition due to a hazard in a nonoscillating circuit

One point we would like to stress is the fact that only race conditions at nodes that are the gates of transistors can propagate into other transistor groups. We call such nodes controlling nodes. All other nodes are inessential nodes. Note that practically every circuit has race conditions at inessential nodes. For example, in the 2-input NAND of figure 3 (which is the same circuit as the one in Figures 1 and 2), there is a race condition at node c when the inputs switch from 11 to 00. But this fact is not of importance, since the output of the NAND is node d and hence c is an inessential node, and not a controlling node.
In the following sections, we develop a linear-time algorithm for the detection of race condition at nodes in any nonoscillating circuit with unit fanout for transistor groups, and use it as a basis for a fast and reliable switch-level simulator for such circuits. But first, we prove the following result on race conditions at strong nodes.

**Lemma 1:** In a nonoscillating circuit in which no controlling node has level $x$, a race condition can never occur at a strong node, or at any node connected to a strong node through a sequence of source-drains of transistors that are on.

**Proof** Since the circuit is nonoscillating, there is no feedback. By assumption, the inputs are stable, so a race condition does not occur at any of these nodes. Let $n$ be a strong node in the $i$th transistor group. Since there is no feedback in this group, $n$ gets a race condition iff different sequences of switchings of the gates of transistors in this component can give rise to different final levels at $n$. However, since no gate has level $x$, the final configuration of on and off transistors in this group is uniquely specified and is independent of the sequence of switchings of the gates. The final value at $n$ is a strong value, since $n$ is a strong node. This strong value must be its unique final value.
value, since it is either the intrinsic value of a strong node, or an undefined value induced by the merge of two nodes of the same strength and different levels. In either case, the value is not affected by the sequence of switchings of the gates.

Similarly, if a node $n'$ is not strong, but is connected to a strong node through a sequence of source-drains of transistors that are on, then it acquires the level of the strong node. Since we have shown above that a strong node cannot have a race condition, neither can $n'$. This proves the lemma.[]

The rest of the paper is organized as follows: In Section 3, we derive a fast race detection algorithm for nonoscillating circuits which will stop if a controlling node ever settles at an undefined or unknown level. By the above lemma, a race condition can occur only at weak nodes. In Section 4, we derive a more general algorithm to handle unknown and undefined levels at controlling nodes. The algorithms we derive assume that all inputs that can change are gated through a clock, so that switching of external signals occurs only in the clocked mode. So, for instance, we cannot have the situation of figure 4 where $I_1$ and $I_2$ are external signals. This is the basis of synchronous circuits. In Section 5, we illustrate the use of our algorithm by an example, and in Section 6, we investigate the effect of fanout from transistor groups to establish that the problem of race detection in nonoscillating circuits becomes NP-complete if a fanout of two or more is allowed for transistor groups. The paper concludes with a summary and some open questions in Section 7.

![Figure 4]

Inadmissible configuration of external signals to a circuit
3. The Basic Algorithm

In Section 2 we pointed out that under the LSV simulation, the transistor groups are ordered so that the switching of a node in transistor group $i$ does not influence any node in group $i-1$ or less. Such an ordering can be found in linear time in any nonoscillating circuit by topologically sorting the nodes of the control graph. In this section, we also assume that the algorithm terminates any time a gate has an unknown or undefined level. This means that a race condition occurs at node $n$ in transistor group $i$ if and only if two different sequences of switchings of the gates of transistors whose source-drains are in this group, cause node $n$ to settle at different final levels. This immediately suggests an algorithm to detect the presence of a race condition at any node $n$ in transistor group $i$: If there are $k$ transistors in group $i$ whose gates changed level at some time during this phase, then, for each of the $k!$ different ways these gates can switch, determine the final level at node $n$. If the level is the same in all cases, then there is no race condition at node $n$, otherwise there is. (Actually, the number of different sequences of gate switchings will be greater than $k!$ if there are hazards. However, once the switching pattern of the gates is known, the number of different sequences is known, and is finite.)

The algorithm described above is clearly not fast, though its running time depends on the maximum number of transistor source-drains in a transistor group, and hence is dependent on the structure of the circuit. However, we present below a race-detection algorithm that is linear in the number of transistors. Note once again that, in this section, we make the assumption that the algorithm terminates if the gate of a transistor ever gets an undefined value, or a race condition.

We first introduce the notion of a switched node.

**Definition 1:** A node $n$ is a **switched node** during a given clock phase if it could be at level 0 and at level 1 during the clock phase.

Thus, for instance, a switched node is one that changed its level during the phase; or had a hazard during the phase; or whose level during the phase was $x$ (i.e., $u$ or $w$). The concept of switched nodes is important for two reasons: Firstly, every node with a race condition is
also a switched node, so we can confine our attention to switched nodes in determining nodes with a race condition. Secondly, it turns out that transistor gates that are switched nodes are the critical ones that contribute to race condition in the transistor group they control.

Next, we introduce the concept of a switched graph. Recall that, while preprocessing the circuit, we introduced an edge between the source and drain of each transistor to get an undirected multigraph, whose connected components were the transistor groups. After we have simulated the nodes in transistor groups 1 to i-1, we form the switched graph of group i as follows:

Algorithm 1: Switched Graph

begin

1) Retain edges for only those transistors whose gates are not switched nodes, and are at logical level one. Collapse each connected component of this graph into a single node whose level is the common level of all the nodes in the component, and whose strength is the strength of a strongest node in the component. Call each such node a level-one-macronode.

2) Introduce an edge for each transistor whose gate is a switched node.

end.

The graph obtained by steps 1 and 2 is called the switched graph of the ith transistor group for this clock phase, and is denoted by $G^S_i$. Clearly, $G^S_i$ is a subgraph of transistor group i, and may contain several connected components.

Lemma 2: Every node in a level-one-macronode gets the same value at all points in the clock phase and this is the value obtained by treating the macronode as a regular node in the switch level circuit.  
Proof Trivial.[]

In what follows, we will treat a level-one-macronode as a regular node in the circuit until the final updating of the individual nodes at the end of the clock phase.
Definition 2: The $j$-switched graph $G_{ij}^s$ of the $i$th transistor group is the subgraph of $G_{ij}^s$ that is obtained by deleting all nodes (including level-1 macronodes) of strength greater than $j$, $1 < j < r$.

Note: $G_{ij}^s = G_{ir}^s$.

Our algorithm for race detection consists of two main steps:

1) We first identify the switched nodes in a transistor group using our information on which of the gate nodes are switched nodes.

2) We then identify the subset of switched nodes that have a race condition, while at the same time, determining the unique level for the remaining nodes.

Our next algorithm finds the switched nodes in transistor group $i$, given the switched graph $G_{ij}^s$ and the initial levels of nodes in $G_{ij}^s$. The algorithm uses level $x$ for either $u$ or $w$, and to identify switched nodes. To each node $n$ in $G_{ij}^s$, the algorithm associates a set $L_n$ that contains a set of possible levels for $n$. To start with, $L_n$ contains the initial level of $n$ if $n$ is a weak node, and the intrinsic level of $n$ if $n$ is a strong node. If $|L_n|$ becomes greater than one, then $L_n$ is reinitialized to $x$ to indicate that $n$ is a switched node. At the end of the algorithm, $L_n$ contains $x$ iff $n$ is a switched node. We prove this result in Theorem 1 below. We will refer to the content of $L_n$ at the end of Algorithm 2 as the level set of $n$.

Algorithm 2: Switched Nodes

begin

1) For each weak node $n$ in $G_{ij}^s$, initialize $L_n$ to its initial level. For each strong node $n$ in $G_{ij}^s$, initialize $L_n$ to its intrinsic level.

2) With each connected component $C$ in $G_{ij}^s$ do SWITCHEDNODES($C, r$).

end.

Procedure SWITCHEDNODES($S, k$);
{ $S$ is a connected component of an $k$-switched graph }

begin
If there are two nodes of strength $k$ in $S$ with different levels in their level set, or there is a node of strength $k$ in $S$ with level $x$ in its level set, then for every node $n$ in $S$, reinitialize $L_n$ to $x$; else

begin

a) If there is a node of strength $k$ in $S$ with level $\ell$ in its level set, then $L_n = L_n \cup \{\ell\}$ for each $n$ in $S$.

b) For each node $n$ for which $|L_n| > 1$ do reinitialize $L_n$ to $x$.

c) If $k > 1$, then delete all nodes of strength $k$ from $S$ to obtain a subgraph of $G^S_{i(k-1)}$. With each connected component $C$ in this subgraph of $S$ do SWITCHEDNODES($C, k-1$).

end;

end;

Note: After execution of Algorithm 2, each $L_n$ contains exactly one entry: 0, 1, or $x$.

Theorem 1: Given the switched graph $G^S_i$, and the initial level of all weak nodes in $G^S_i$, Algorithm 2 correctly identifies the set of switched nodes in $G^S_i$, and the unique final level for the remaining nodes.

Proof: We divide the proof into three parts. For a node $n$ in $G^S_i$ with level $\ell_0$ in $L_n$ at the end of Algorithm 2,

a) if $\ell_0 = x$ then we show that $n$ is a switched node;

b) if $n$ is a switched node, then we show that $\ell_0 = x$; and

c) if $\ell_0 = 1$ or $\ell_0 = 0$, then $\ell$ is the unique level at which $n$ stayed throughout.

To prove a), we show that $n$ can take on every level assigned to $L_n$ at some point during the execution of the algorithm. Then, since $\ell_0 = x$ iff $x$ is in $L_n$ at some point or if $|L_n| > 1$ at some point, the assertion follows.

Let $n$ be a node of strength $k$ and initial level $\ell_0$. If $n$ is a weak node, then the initial assignment of $\ell_0$ to $L_n$ certainly corresponds to a level that $n$ takes on. If $n$ is a strong node, then the initial assignment to $L_n$ is the intrinsic level of $n$. Consider the sequence of switchings of edges in $G^S_i$ in which all edges that are on switch off before any
edge switches on. Then at the point just before any edge switches on, n is isolated from all other nodes (since every node is isolated from every other node in \( G_i^s \) at this point) and hence n attains its intrinsic level. This establishes assertion a) for the initialization step.

\( L_n \) can get additional levels in the recursive calls to procedure SWITCHEDNODES corresponding to connected components of j-switched graphs \( G_i^s, G_i^s, G_i^s, \ldots, G_i^s \) (here \( u \leq k \) if \( L_n \) is reinitialized to x at an earlier stage). Consider the level assigned to it during the processing of a connected component S of \( G_i^s \) for some \( m, u \leq m \leq x \). First we consider the case when there is no node of strength m in S whose level set presently contains x. Let E be the set of edges that were initially on, connecting S to nodes in \( G_i^s \) that are not in S. Partition E as \( E = E_{m+1} \cup E_m \cup \ldots \cup E_1 \), where \( E_j \) contains edges connecting a node in S to a node of strength j. Consider the sequence of switchings in which first all edges in \( E_{m+1} \) switch off simultaneously, then all edges in \( E_{m+2}, \ldots \), then all edges in \( E_m \). At this point in this sequence of switchings, nodes in S are isolated from all nodes not in S. Further, this sequence of switchings does not change the level of any weak node in \( G_i^s \) (it had the level of the strongest node that it was connected to, and since it was disconnected from that node last, it retains its level). Thus, regardless of whether m represents a strong or weak strength, at this point in the sequence of switchings, every node of strength m has either the level in its level set (which is its initialized level), or x (this iff there are two nodes in S of strength m with different initial levels that are initially connected).

Next, turn on all edges in S that were initially off. At this point in the sequence of switchings, every edge in S is on, and hence every node in S attains the merge of the strongest nodes in it. So, as prescribed by Algorithm 2, if there are two nodes of strength m in S with different level, every node in S becomes a switched node.

On the other hand, let \( n' \) be a node of strength m in \( G_i^s \) whose level set presently contains x. According to Algorithm 2, every node in S should be a switched node in this case, too. If this x was introduced in \( L_n' \) during initialization, then the above sequence of switchings again establishes that every node in S is a switched node. If not, then \( n' \) obtained x in its level set because the initialization level was 0 (1) but in some \( G_{ij}^s, j > m \), \( n' \) was connected to a node of strength j and level 1 or x (0 or x). But then, every node in S also attained level 1.
or $x$ (0 or $x$) during the processing of $G_{ij}$, while the above sequence of switchings causes every node in $S$ to get level 0 (1). Thus every node in $S$ could get each of level 0 (or $x$) and level 1 at some point during some sequence of switchings. So every node in $S$ is a switched node if some node of strength $m$ in $S$ has $x$ in its level set. This establishes assertion a).

To establish assertion b), we enumerate the various ways in which a node $n$ of strength $k$ can become a switched node:

i) $n$ initially had level $x$.

ii) $n$ was connected to a stronger node of strength $k'$ with level $x$ through some sequence of switchings of transistors.

iii) $n$ was connected to another node of same strength and different level through some sequence of switchings of transistors.

iv) It was possible for $n$ to get level 0 at some point in a sequence of switchings and level 1 at some point in another sequence of switchings.

If i) holds, and $n$ is a weak node, then $L_n$ gets assigned $x$ during initialization (note that once $L_n$ gets assigned $x$ at some point during the algorithm, it will continue to have $x$ from that point on). If $n$ is a strong node, then it had initial level $x$ because it was initially either connected to a stronger node of strength $k'$ with level 0 or it was connected to another node of same strength and different level. The latter case is detected during execution of $\text{SWITCHEDNODES}(S, k)$ while the former case is detected during the execution of $\text{SWITCHEDNODES}(S', k')$ in Algorithm 2, where $S$ is the connected component of $G^S_{ik}$ in which $n$ lies, and $S'$ is the connected component of $G^S_{ik'}$, in which $n$ lies.

If ii) holds, then this is detected during the execution of $\text{SWITCHEDNODES}(S', k')$ in Algorithm 2.

If iii) holds, then this is detected during the execution of $\text{SWITCHEDNODES}(S, k)$ in Algorithm 2.

Suppose iv) holds. Then $n$ had initial level 0 (1) and there is a sequence of switchings of transistors such that at some point during that sequence, $n$ was connected to a node $n''$ of strength $k'' > k$ and level 1 (0). Then, in Algorithm 2, during the execution of $\text{SWITCHEDNODES}(S'', k'')$, $n$ will be assigned level $x$, where $S''$ is the connected component of $G^S_{ik''}$, in which $n$ lies. This proves assertion b).
Thus we have established that a node is a switched node iff its level set is assigned x by Algorithm 2. Now, if a node is not switched, then it has the same level throughout the clock phase. This must clearly be its initial level, and it must also be the intrinsic level for strong nodes. In Algorithm 2, the level set of each node is initialized to one of these levels depending on the strength of the node. If the level set of a node gets any more entries the level set is reinitialized to x. Hence, according to Algorithm 2, any node that is not a switched node is assigned either its intrinsic level (if it is a strong node) or its initial level (if it is a weak node). This establishes c).

**Theorem 2:** The time complexity of Algorithm 2 is $O(T_r)$, where $T$ is the number of transistors in $G^S_i$.

**Proof:** The initialization requires $n_i$ units of time, where $n_i$ is the number of nodes in the $i$th transistor group, and this is clearly $O(T)$. Each call to the recursive procedure SWITCHEDNODES(S,k) can be done in $O(S_T)$ time, where $S_T$ is the number of edges in $S$. Procedure SWITCHEDNODES is called at most once for each connected component of $G^S_{ij}$, $j=1,2,...,r$. Since the connected components for a fixed $j$ are disjoint, and each $G^S_{ij}$ is a subgraph of $G^S_i$, the overall complexity of the recursive calls is $O(T_r)$.

**Note:** Since $r$, the number of different strengths, is a (technology-dependent) constant, Algorithm 2 runs in time linear in the size of the input.

Next, to identify the subset of nodes that have a race condition, we look at the subgraph of the switched graph that contains only edges that correspond to transistors that can be on at the end of the phase. This refers to transistors whose gates have final level 1. We call this graph the on-graph of transistor group $i$ for this clock phase, and denote it by $G^O_i$. We still treat level-one-macronodes as regular nodes. The on-graph may again contain many connected components. Let $C$ be any connected component of the on-graph. From Lemma 1 we know that there can be no race condition at any node in a connected component that contains a strong node. So we only need look at connected components that contain only weak nodes for a possible race condition. For such connected components, we have the following theorem:
Theorem 3: Let $C$ be a connected component in $G^0$, and let $k$ be the strength of a node $n$ of maximum strength in this component. If $k \leq q$ (i.e., all nodes are weak), then the final level of all nodes in $C$ is precisely the level in $L_n$.

Proof: The final level of all nodes in $C$ is determined by the nodes of strength $k$ in $C$. If $L_n$ does not contain $x$, then $n$ is a node of strength $k$ that remained at a single level throughout the clock phase. Hence its final level and the final level of all nodes in $C$ is precisely the level in $L_n$. For the case when $L_n = x$, we can use a method similar to that used in establishing assertion b) in the proof of Theorem 1: For each of the different ways by which $n$ can become a switched node, we can give a sequence of switchings such that $n$ retains that level at the end of all switchings. This establishes the result. The details of the proof are omitted.[]

We use the above theorem in the following algorithm for detecting races. The output from Algorithm 2 is taken as input.

Algorithm 3: Final Levels (no $x$ at gates)

begin

1) Form the on-graph $G^0_i$ by deleting those edges in $G^5_i$ that correspond to transistors whose gates are not at level 1 at the end of the clock phase.

2) For each connected component $C$ of $G^0_i$. If $C$ has a strong node then the final level of all nodes in $C$ is the merge of the intrinsic levels of all nodes in $C$ of maximum strength else the final level of all nodes in $C$ is the level in $L_n$, where $n$ is a weak node of maximum strength in $C$.

end.

We now have the tools to give the restricted simulation algorithm with race detection in a nonoscillating circuit whose nodes have been topologically sorted. We assume that we have pointers so that we can extract each transistor group by turn. Let $N$ be the number of transistor groups in the circuit.
Algorithm 4: Restricted Simulation Algorithm with Race Detection

begin

1) Set input nodes to their new levels, and clocks to correspond to the new phase. Mark all input nodes that changed from their previous level as switched nodes.

2) For i=1,2,...,N, process transistor group i:
   a) Form the switched graph $G^S_i$ using Algorithm 1.
   b) Use Algorithm 2 to detect the presence of switched nodes in $G^S_i$.
   c) Apply Algorithm 3 to detect race conditions, and to find the final level at each node.
   d) Decompose each level-one-macronode into its component nodes, and assign the final level of the macronode to each of the nodes. If the macronode is a switched node, mark each component node as a switched node.
   e) If a controlling node gets final level x, then flag the error and stop.

end.

4. The General Simulation Algorithm

In this section, we extend Algorithm 4 of the previous section to handle the case when the gate of a transistor can be at level x. We need to reexamine the results of the previous section under this additional feature. First, note that we can obtain the level w in another way. Consider the situation in figure 5. The gate of the transistor (node a) has level x, so the transistor can be either on or off in the steady state. If a is at 1, then node c will be at level 0; if a is at 0, then c will be at level 1. Hence c can have two different levels, and so is at level w. If the x at a is caused by a race condition, then the unknown level at c represents a race condition. However, if the x at a is an undefined level, then the unknown level at c is not really a race condition, since it is not caused by differing signal delays on different paths to c. But
since the effect is the same in both cases, we will ignore the difference, and use the term 'race condition' to mean the level $w$. Note, also, that when we allow level $x$ at a controlling node, we can have a race condition at a strong node, and hence Lemma 1 does not hold in this case.

Fortunately, however, Theorems 1 and 2 so hold for this case, since a gate with level $x$ is a switched node by Definition 1. This means that Algorithm 2 goes through without change.

The main addition we need to make is in the evaluation of the final levels. Now we may have transistors in the $i$th transistor group whose gates are at level $x$, so at the end of the clock phase, these transistors may be either on or off. This, of course, is what gives rise to race conditions in strong nodes. We will now develop the tools to evaluate the final levels at nodes when controlling nodes are allowed level $x$. First, we construct the final graph in Algorithm 5. The algorithm takes the output from Algorithm 3 as input.

Algorithm 5: Final Graph

begin
1) Collapse each connected component of the on-graph into a single node that has the common final level as determined by Algorithm 3, and the strength of a strongest node in the component. Call each such node a final-macronode.

2) Introduce an edge for each transistor whose gate has level $x$.

We denote the final graph of the $i$th transistor group derived in Algorithm 5 by $G_i^f$. By analogy with Definition 2, we define the $j$-final graph $G_{ij}^f$:

**Definition 3:** The $j$-final graph $G_{ij}^f$ of the $i$th transistor group is the subgraph of $G_i^f$ that is obtained by deleting all nodes (including final-macronodes) of strength greater than $j$, $1 < j < r$.

Note that Algorithm 5 is very similar to Algorithm 1, except that in step 1, we use transistors whose gates have final level 1, rather than transistors whose gates are at level 1 all through the clock phase; and in step 2 we use transistors whose gates have final level $x$, rather than transistors whose gates are switched nodes. We can modify Lemma 1 to obtain the following:

**Lemma 2:** Every node in a final-macronode of Algorithm 5 gets the same final level, and this is the level obtained by treating the macronode as a regular node in the circuit.

Next, we have the interesting result that each of Theorems 1 and 2 translates in a very direct way into a corresponding theorem about the final levels of nodes in the final graph. We state the new theorems below, but we omit the proofs, since they can be obtained directly from the proofs of the earlier theorems.

**Theorem 4:** Given the final graph $G_i^f$ and the level set of all nodes, Algorithm 2 when implemented on $G_i^f$ rather than $G_i^s$ correctly identifies the set of nodes with race condition and the unique final level for the remaining nodes.

**Theorem 5:** The time complexity of Algorithm 2 when implemented on $G_i^f$ is $O(Tr)$ where $T$ is the number of transistors in $G_i^f$. 
Since Theorems 4 and 5 have an exact correspondence with Theorems 1 and 2, the algorithm to determine the final levels corresponds in a direct way to Algorithm 2. However, we will not need the full power of Algorithm 2 because we need to check for race conditions only at strong nodes. For weak nodes, the levels obtained in Algorithm 3 are the final levels, as shown in the following theorem:

**Theorem 6:** Let C be a weak node or macronode in the final graph, whose level from Algorithm 3 is 1. Then, this level does not change if some of the transistors that are not on have level x at their gates.

**Proof:** From Theorem 3, a weak macronode gets a race condition if it is a switched node, and retains its unique level otherwise. We also know that if a node has a race condition, then it is switched node. Since gates at level x are also switched nodes, no node can get a race condition unless it is a switched node. Hence the final level for a weak macronode obtained from Algorithm 3 remains its final level even in the presence of x at some gates.

From Theorem 6, we have the result that, given the output from Algorithm 3, we only need to determine the final levels at strong nodes. This is done in Algorithm 6 below, which applies Algorithm 2 to the final graph, but restricts the recursive calls to strong nodes.

**Algorithm 6: Final Levels at Strong Nodes with Race Detection**

```
begin
  1) For each strong node or macronode n in G^f_i, initialize L_n to its intrinsic level.
  2) With each connected component C in G^f_i do RACENODES(C, r).
end.
```

Procedure RACENODES(S, k);
[S is a connected component of a k_final graph]

```
begin
  If there are two nodes of strength k in S with different levels in their level sets, or there is a node of strength k in S with level x in its
```
level set, then for every node in S, reinitialize $L_n$ to $x$; else
begin
a) If there is a node of strength $k$ in S with level $l$ in its level
set, then $L_n = L_n \cup \{l\}$ for each $n$ in S.

b) For each node $n$ for which $|L_n| > 1$, reinitialize $L_n$ to $x$.

c) If $k > q$, then delete all nodes of strength $k$ from S to obtain a sub-
    graph of $G_i^{f(k-1)}$. With each connected component $C$ in this subgraph
    of S do RACENODES(C, k-1).
end;

end;

We now give the general simulation algorithm for nonoscillating
circuits. As before, $N$ will denote the number of transistor groups in
the circuit.

Algorithm 7: Simulation Algorithm with Race Detection for Nonoscillating
Circuits

1) Set input nodes to their new levels, and clocks to correspond to
   the new phase. Mark all input nodes that changed from their previous
   level.

2) For $i=1,2,\ldots,N$, process transistor group $i$:
   a) Form the switched graph of transistor group $i$ using Algorithm 1.
   b) Use Algorithm 2 on the switched graph to find the switched nodes
      in transistor group $i$ and the unique final level for the remaining
      nodes.
   c) Apply Algorithm 3 to detect race conditions at weak nodes, and
      to find the final level at each node, assuming all transistors with
      $x$ at their gates are off.
   d) Form the final graph of transistor group $i$ using Algorithm 5.
e) Apply Algorithm 6 to detect race conditions at strong nodes and their final levels.

f) Decompose each final-macronode and set the level of each component node equal to the final level of the macronode.

g) Decompose each level-one-macronode into its component nodes, and assign the final level of the macronode to each of the nodes. If the macronode is a switched node, mark each node as a switched node.

end.

Theorem 7: Algorithm 7 runs in $O(Tr)$ time, where $T$ is the number of transistors in the circuit.

Proof We have shown in Theorems 2 and 5 that for each $i$, step 2 requires $O(T_i^r)$ time, where $T_i$ is the number of transistors in the $i$th transistor group. Since the transistor groups are disjoint, this gives an overall time complexity of $O(Tr)$.

5. An Example

In this section, we will illustrate the use of Algorithm 7 by simulating one clock phase on the transistor group of figure 6. The circuit models nMOS, and hence the merge rules are from Table 1.

Step 1) Each of nodes $l$ to $r$ is assumed to be an input node.

Node ordering = $l$, m, n, o, p, q, r, a, b, c, d, e, f where nodes a, b, c, d, e, and f are in transistor group 8, and each of the previous nodes is in a separate group by itself.

Initial levels at the gates are $l=0$, $m=1$, $n=1$, $o=1$, $p=0$, $q=0$, $r=0$.

Initial levels at the nodes in group 8 are $a=0$, $b=0$, $c=0$, $d=1$, $e=0$, $f=0$.

The new levels at the gates are $l=x$, $m=x$, $n=x$, $o=x$, $p=1$, $q=1$, $r=0$.

The switched gates are n, o, p, and q.

Step 2a) The switched graph is given in figure 7.

Step 2b) Use Algorithm 2 on connected component C containing nodes a, b, c, d, and e. Node e is an input node at level 0, hence add 0 to level
The switched graph for the example

set of all nodes. Node a is a pullup node, hence add 1 to level set of all nodes except e. Level-one-macronode bc is a weak node at 0, while node d is a weak node at 1, hence add level u to the level sets of d and e. Hence the switched level sets are:

node a: \{0,1\};
node bc: \{0,1,u\};
node d: \{0,1,u\};
node e: \{0\}.

Mark nodes a, bc, and d as switched nodes.
Step 2c) The on-graph is given in figure 8.

Figure 8
The on-graph for the example

Use Algorithm 3 on connected component C containing nodes b, c, and d.
Final level set for bc and d is \{0,1,u\}.
Final level for a is 1, and for e and f, 0.

Step 2d) The final graph is given in figure 9.

Figure 9
The final graph for the example

Step 2e) Use Algorithm 6 on connected component C containing nodes a, bcd, and e. Node e is an input node at level 0, so a can be at level 0. Hence final level set for a is \{0,1\}, while for e, it remains 0.

Step 2f) Final level set for nodes bc and d is \{0,1,u\}.
Step 2g) Final level set for nodes b and c is \( \{0, 1, u\} \). Mark nodes b and c as switched nodes.

There is a race condition at nodes a, b, c, and d during this clock phase. The set of switched nodes is also the same.

If nodes n and p had switched to 0 rather than to x, then the final level set of a becomes just 1, so that there is no race condition at a during the phase. However, a still remains a switched node by Step 2b.

6. Fanout Considerations

In Sections 3 and 4, we analyzed carefully, the production of races in a transistor group, and we presented a provably correct algorithm to detect race conditions in a transistor group, given the initial level at nodes in the group, and information on the levels at the gates. Our only reference to the control graph was the presence of a topologically-sorted ordering for the transistor groups which was possible because the control graph is acyclic.

In this section, we probe more carefully into the structure of the control graph and its effect on the correctness of Algorithm 7. We first show that the problem of race detection in a nonoscillating circuit is NP-complete if unbounded fanout is allowed from transistor groups. We then refine our reduction to show that the problem remains NP-complete even if only a fanout of two is allowed for transistor groups and each node in the circuit is gate to no more than one transistor.

We present our first NP-completeness result below:

**Problem A: Race Detection in Nonoscillating (i.e., Acyclic) Transistor Switch-Level Circuits**

**Instance:** A nonoscillating transistor switch-level circuit, initial values at all nodes, new values for the input nodes, and an observed node n in the circuit.

**Question:** Does the new set of values for the input nodes cause a race condition at n?
NP-Completeness Proof We will use a reduction from SATISFIABILITY [GaJo79].

Assume we are given an arbitrary instance of SATISFIABILITY: a set \( U \) of variables and a collection \( C \) of clauses over \( U \) (see [GaJo79] for an explanation of this specification). We construct a boolean acyclic circuit for this instance of SATISFIABILITY and from it we derive a switch-level acyclic circuit for this instance that contains a number of transistors linear in the size of the instance. Figure 10 shows a general boolean acyclic circuit to implement a given instance of SATISFIABILITY. Note that all gates have a fanout of one, and hence, the corresponding switch-level circuit will have a fanout of one for transistor groups. We denote this switch-level circuit by a black box \( B \). The box \( B \) has \( |U| \) inputs and one output; the output gives the truth value of the boolean expression corresponding to the input levels. We make the output node, the node \( n \) in an instance of Problem A.

![Figure 10: Boolean gate implementation of an instance of SATISFIABILITY](image_url)

Let \( U = \{A_1, A_2, \ldots, A_k\} \) be the set of variables. We generate each \( A_i \) by our 'race-production' circuit of figure 1, which we repeat below in figure 11 in a more general setting: Node \( y_i \) is a strong node of strength \( S_r \) and intrinsic level 0. Node \( x_i \) is a strong node of strength \( S_{r-1} \) (recall that \( S_r > S_{r-1} \)) and intrinsic level 1. If the levels at gates \( a_i \) and \( b_i \) switch from 1 to 0 in the same clock phase, then a race condition is induced at \( A_i \), for each \( i = 1, 2, \ldots, k \).
Our instance of Problem A is the circuit shown in figure 12.

The input nodes are $x_i, y_i, a_i, b_i, i=1, 2, \ldots, k$. Note that each $x_i$ will be assigned level 1 and each $y_i$ will be assigned level 0 in each clock phase, since the $x_i$ and $y_i$ are strong nodes with intrinsic levels 1 and 0 respectively. The initial assignment of levels to all $a_i$ and $b_i$ is 1. This gives level 0 to all variables $A_i$. We compute the levels at all controlling nodes in B and consequently at $n$ and specify this as part of the input. We assign level $x$ to weak nodes that are not connected to a strong node with this assignment of input levels. This specifies the initial levels at all nodes in the circuit. We specify the new levels...
for the $a_i$ and $b_i$ as all zeros. This is our instance for Problem A, and the question we are required to answer is whether node $n$ has a race condition.

Our input specification causes a race condition for each $A_i$, so each of the $k$ variables can independently take on one of the levels 0 or 1. Now, assume Problem A can be solved in polynomial time. Then, we claim that we can use that polynomial time algorithm on the above instance to obtain a solution to the SATISFIABILITY problem as follows: We look at the initial level at $n$. This corresponds to the evaluation of the boolean expression when all variables are set to 0. If the level at $n$ is 1, then the instance is satisfiable and we are done. If not, and the level at $n$ is 0, then we use the polynomial-time algorithm for Problem A to determine if $n$ has a race condition with the new input values. Because we have induced an independent race condition on each of nodes $A_1$ to $A_k$, each of the $k$ variables could independently be either at level 0 or at level 1. If the algorithm determines that $n$ does not have a race condition, then all possible assignment of zeros and ones to the $A_i$'s must cause a unique level at node $n$, i.e., the level 0 since this is the level at $n$ when all $A_i$'s are 0. Thus the underlying instance is not satisfiable if $n$ does not have a race condition. On the other hand, if there is a race condition at $n$, then there is some assignment of zeros and ones to the $A_i$'s for which the level at $n$ is 1, i.e., the instance is satisfiable. Hence there is a race condition at $n$ iff the given instance of SATISFIABILITY is satisfiable.

Hence Problem A is NP-complete.[]

We have shown in Sections 3 and 4 that, given full information on the levels at gates (and the switched nodes among them) of a transistor group, we can determine in linear time all information on race conditions at nodes in the transistor group. The reason why the general problem is still NP-complete is due to the presence of fanout from a transistor group, i.e., the out-degree of a node in a control graph may be large. Suppose $n$ is a node in a circuit that is gate to $k$ transistors, and $n$ has a race condition, i.e., $n$ can be either at level 0 or at level 1. Then, to correctly characterize the effect of this on the levels at other nodes in the circuit, we need to explore the case when all $k$ transistors are on and the case when all $k$ transistors are off. But our algorithm assigns level $x$ to each of the $k$ gates, and looks at each of them in isolation. In the process, it may identify a race condition
at some node. But this race condition could occur only if, say, the ith transistor controlled by n is on, and the jth transistor controlled by n is off. Such a situation can never occur since either all transistors controlled by n are on, or all of them are off.

It is possible that, in the circuit of figure 11, node $a_i$ switches off first and then node $b_i$, but the lag between the two is such that node $A_i$ is not able to charge up completely to level 1. In this case, we will have an undefined level at $A_i$ (as distinct from an unknown level). It is also possible that the different transistors controlled by $A_i$ have different voltage thresholds at which they switch on or off. In that case, an undefined level at $A_i$ may be interpreted as a 1 by one transistor and a 0 by another. We have assumed that this is not possible. If such a behavior is possible in a circuit, then our NP-completeness result does not hold, because we cannot assume that a race condition at a controlling node causes transistors controlled by it to be all on or all off.

Our earlier reduction shows that race detection in nonoscillating circuits is NP-complete if unbounded fanout is allowed for transistor groups. It is known that SATISFIABILITY remains NP-complete even if there are at most 3 clauses in C that contain $A_i$ or its complement, for each i. This implies a stronger result that race detection in nonoscillating circuits is NP-complete if fanout of three or more is allowed for transistor groups. We can refine this still further to show that race detection in nonoscillating circuits if NP-complete even if fanout is restricted to two for any transistor group, and no node in the circuit is gate to more than one transistor.

We use the construction of figure 13. The basic unit is the black box $B'$ that inverts the input variable using one transistor and appropriate node strengths, and a pass transistor at the output whose gate is fixed at level 1 to obtain a fanout of two from the transistor group and a fanout of no more than one for any node within the group. Each output of $B'$ is made an input to another $B'$, so that we obtain four nodes with level $A_i$ using six transistors, and complying with the fanout restrictions. This circuit is sufficient to properly buffer to each input of a circuit implementing an instance of SATISFIABILITY in which there are at most 3 clauses that contain an occurrence of any variable or its complement.
Construction to limit fanout in the circuit of figure 12

The circuit we have constructed for our NP-completeness reduction is robust in the sense that it can be built in nMOS and will function correctly, even after taking into account questions like threshold voltage drops and area of pullup transistors (which can be made to be small using a NOR-NOR construction).

7. Conclusion

In this paper, we have presented a fast and reliable switch-level simulator for a class of nonoscillating transistor switch-level circuits. We have proved that our linear-time algorithm works correctly when transistor groups have unit fanout. We have also shown that the problem is NP-complete if fanout of two or more is allowed for transistor groups. However, if we run our algorithm on such circuits, we will still detect all race conditions; in addition, however, the algorithm may incorrectly specify a race condition at a node n when, in fact, there really is no race condition at n. That is, Algorithm 7 performs over-cautiously when run on general nonoscillating circuits. It appears that an algorithm of this type is much preferable to an algorithm that may not detect a race condition that exists, as in [Bry80], [LiSeVa81], [Bry84]. Given the NP-completeness result for race detection in general nonoscillating circuits, our over-cautious linear-time algorithm appears to be the best that we can hope for.
Recently, [Bry83] has introduced ternary simulation in MOSSIM II and found it to work well. Ternary simulation is a race detection technique used in the context of logic gate simulation ([Ei65], [BrzYo79]), and has some similarity to the technique presented in this paper. However, no attempt was made in [Bry83] to prove the validity of applying ternary simulation to switch-level circuits. In this paper, we have presented a method that has a certain similarity to ternary simulation, but all of our assertions and algorithms are supported by proofs. Also, by using a completely graph-theoretic formulation, we are able to derive clean time-complexity results. In [Bry84], for instance, the author starts out with a graphical formulation, but resorts to sparse matrix manipulation to obtain the final result, making it difficult to obtain a linear time bound. (This algorithm does not detect race conditions.)

While it is true that race conditions can occur in unexpected ways in switch-level circuits, it is also true that most MOS circuits using structured design methodology (e.g., as in [MeCo80]) do not have race conditions. We have developed some simple and general design rules that guarantee race-free behavior in a circuit, and a very fast simulator that tests such designs [Ra83a].

The algorithms presented in this paper can be considerably simplified when applied to nMOS. The corresponding simplified algorithms for the merge rules of Table 1 can be found in [Ra83b].

An obvious extension to this work is to develop fast race detection algorithms for circuits with feedback. We have some preliminary results in this direction.

Acknowledgements: I would like to acknowledge useful discussions with Andrea LaPaugh and Ken Steiglitz at Princeton University.

References


