Boolean Analysis of MOS Circuits*

Randal E. Bryant
Computer Science Department
Carnegie-Mellon University
Pittsburgh, PA 15213

February, 1987

Abstract

The switch-level model represents a digital metal-oxide semiconductor (MOS) circuit as a network of charge storage nodes connected by resistive transistor switches. The functionality of such a network can be expressed as a series of systems of Boolean equations. Solving these equations symbolically yields a set of Boolean formulas that describe the mapping from input and current state to the new network state. This analysis supports the same class of networks as the switch-level simulator MOSSIM II and provides the same functionality, including the handling of bidirectional effects and indeterminate (X) logic values. In the worst case, the analysis of an n node network can yield a set of formulas containing a total of $O(n^3)$ operations. However, all but a limited set of dense, pass-transistor networks give formulas with $O(n)$ total operations. The analysis can serve as the basis of efficient programs for a variety of logic design tasks, including: logic simulation (on both conventional and special purpose computers), fault simulation, test generation, and symbolic verification.
Keywords and phrases: switch-level networks, symbolic analysis, logic simulation, fault simulation, simulation accelerators.

1 Introduction

The switch-level model [1] has proved successful as an abstract representation of digital metal-oxide semiconductor (MOS) circuits for a variety of applications. This model represents a circuit in terms of its exact transistor structure but describes the electrical behavior in a highly idealized way. It expresses transistor conductances and node capacitances by discrete strength and size values; represents node voltages by discrete states 0, 1, and X (for invalid or indeterminate); and makes no attempt to model exact circuit timing. The switch-level model can capture many of the important phenomena encountered in MOS circuits such as: ratioed, complementary, and precharged logic; dynamic memory; and bidirectional pass

*This research was supported in part by the Defense Advanced Research Projects, Agency ARPA Order Number 4976, and in part by the Semiconductor Research Corporation under Contract 86-01-068.

transistors. Unlike programs that attempt to model circuits at a detailed electrical level, programs based on the switch-level model can operate at speeds approaching those of their counterparts based on more traditional gate-level models. Examples of applications that have successfully applied switch-level models include logic simulators [1, 2], fault simulators [3, 4], test pattern generators [5, 6], and symbolic verifiers [7, 8].

1.1 Switch-Level Algorithms

Most programs that model circuits at the switch level utilize totally different algorithms than those developed for logic gate circuits. To accommodate the bidirectional nature of the transistors, they compute the state of a node by applying graph algorithms to trace the connections between nodes formed by conducting transistors. This departure from tradition has several drawbacks. First, considerable effort is often required to adapt existing techniques for use at the switch-level. For example, in implementing the fault simulator FMOSSIM, we found it quite challenging to adapt concurrent simulation techniques [9], although the resulting performance proved well worth the effort. Similarly, automatic test pattern generation for switch-level circuits has not yet reached the success achieved for logic gate circuits. Second, although programs based on the switch-level model have reasonable performance, they fall short of those based on gate-level models. Computing node states by applying graph algorithms to the transistor data structure requires significantly greater effort than computing the output of a logic gate. Finally, these algorithms do not map well onto the special purpose processors that have been developed to accelerate such tasks as logic gate simulation [10, 11, 12]. Although special purpose processors for switch-level simulation have been designed and constructed [13, 14], these processors require a fair amount of specialized hardware. It is unlikely they will ever achieve the cost/performance of processors that support only gate-level evaluation.

1.2 A New Approach

This paper proposes a new approach that deals with all aspects unique to the switch-level model in a preprocessing step. The preprocessor "compiles" a switch-level network into a set of Boolean formulas. For each node, a pair of formulas specifies its steady state response as a function of the initial node states. A simulator can then compute new node states by simply evaluating the appropriate formulas. Fault simulators and test generators can utilize traditional techniques by treating the set of formulas like a logic gate network. The formulas can be translated directly into machine language instructions for fast evaluation on a general purpose computer, or they can be mapped onto any special-purpose hardware that supports Boolean evaluation. An efficient symbolic analyzer, the subject of this paper, serves as the basis of this preprocessing.

This approach has advantages over traditional methods of switch-level evaluation in terms of both speed and flexibility. As an analogy, a programming language compiler yields a performance advantage over an interpreter, because the cost of translating the program into machine instructions is paid only once during compilation rather than repeatedly during execution. Similarly, the analyzer gives a performance advantage over traditional switch-level algorithms, because the added cost of switch-level evaluation is paid only once during preprocessing. In contrast to special purpose hardware for switch-level evaluation, many extensions to the model can be made by simply modifying the analyzer, a much simpler task than modifying the hardware. Furthermore, the Boolean description of switch-level subnetworks can more easily be combined with subnetworks modeled at other levels for mixed-mode evaluation. Finally, as will be discussed briefly, the preprocessor generates a description that can be executed with a far greater degree of parallelism than is possible with more conventional switch-level algorithms.

The analyzer described in this paper supports the same class of switch-level networks as the simulator MOSSIM II [1]. It captures all aspects of the MOSSIM II model including: bidirectional effects, different signal strengths, and indeterminate (X) logic values. The analysis of an n node network produces a set of formulas with a total of at most $O(n^3)$ operations. For all but a very small class of dense, pass transistor networks (e.g., barrel shifters), at most $O(n)$ operations are required. Hence, for practical purposes this approach incurs the same asymptotic complexity as other switch-level programs.

1.3 Related Work

Other researchers have developed preprocessors to translate a switch-level network into some algebraic representation that allows efficient evaluation. These previous efforts had, for the most part, limited generality and accuracy. In addition, they did not achieve acceptable efficiency. Pfister of IBM [15] probably deserves credit for originating the idea of describing arbitrary MOS circuits in terms of Boolean operations. He was seeking a way to perform switch-level simulation on the Yorktown Simulation Engine (YSE) [10].

Researchers at IBM [16] have modified and adapted a traditional switch-level algorithm for execution on the YSE. Their approach can be viewed as generating code to iteratively solve a system of equations in an algebra where elements encode both the strength and the state of a signal [17]. To accommodate the small word size of the machine, they restrict the number of signal strengths to 3 and use a pessimistic method for computing the effects of unknown states. More seriously, since the machine cannot perform data dependent branches, their code must always iterate a worst case number of times. For many transistor structures with n nodes and t transistors, this requires a total of $O(n\,t)$ YSE instructions, a high cost in both space and time. In a related effort, the SLS program developed at IBM [18] generates code for a general purpose computer that executes a single iteration in the solution of the same system of equations solved by MOSSIM II. During simulation this code is executed repeatedly until the values converge. Although this program achieves impressive performance on a variety of circuits, a significant class of pass transistor networks can require many iterations to converge. Furthermore, this approach cannot be implemented on existing simulation hardware, nor can it aid such tasks as test pattern generation or symbolic verification.

Others have attempted to express switch-level algorithms in terms of either Boolean or closely related algebras. All of these efforts have yielded highly inefficient results—in the worst case the size of the algebraic description can grow exponentially with the size of the network. These programs partition the circuit into subnetworks and analyze each subnetwork separately. Most subnetworks are quite small—containing no more than 10 transistors.

Hence even an exponential algorithm can have practical value. However, we have often encountered circuits containing subnetworks of 1000 or more transistors. For such cases these algorithms would be totally inadequate. The method developed by Cerny and Gecsei [19] creates a symbolic representation of all possible partitionings of a subnetwork into connected components formed by the conducting transistors. All but the smallest subnetworks have many partitionings, and hence this approach has limited potential. The methods of Ditlow, *et al*, [20] of Hajj and Saab, [21] and of Terman [2] enumerate the set of all simple paths to each node and then encode information about each path algebraically. For many pass transistor networks, (e.g., the Tally circuit of Mead and Conway [22]), the number of such paths grows exponentially with the number of transistors. Furthermore, all of these methods place more restrictions on the class of networks than does MOSSIM II, and some do not do as well at modeling the effects of X values. Finally, the method of Hajj and Saab utilizes a mixed Boolean-integer algebra, in which the 1's in different sets of Booleans must be tabulated and compared. Such an algebra seems needlessly awkward and would be hard to implement on most simulation hardware.

In a different application of symbolic analysis, the MOSSYM program [7] simulates MOS circuits *symbolically*. A symbolic simulator resembles a conventional simulator except that the input patterns may consist of Boolean variables in addition to the constants 0 and 1. The node states computed by the simulator represent Boolean functions over the present and past input variables. This program is designed to rigorously verify digital circuits, proving their correctness for all possible input sequences. As a consequence, it must solve Boolean equivalence, a well-known NP-hard problem [23]. The worst case performance of the program is exponential in the number of variables, and many researchers believe no better performance can be achieved. The methods used by MOSSYM for computing the Boolean behavior of a switch-level network form the basis of the analyzer described here. However, the analyzer can use different data structures and algorithms for representing Boolean functions, since it need not prove equivalence. Consequently, while the simplification algorithms may not yield the most compact formulas possible, they have acceptable worst case performance. Future versions of MOSSYM will operate on preprocessed networks rather than on the transistor structure directly, gaining the same benefits from preprocessing as do more conventional simulators.

1.4 Overview

The analyzer presented in this paper overcomes many weaknesses of the previous attempts. Important features include:

- It partitions the network into channel-connected subnetworks and derives the steady state response of each subnetwork separately. This partitioning divides the analysis task into smaller subproblems.

- It encodes logic states 0, 1, and X with pairs of Boolean values. By this encoding, it can accurately characterize the effects of unknown node and transistor states with Boolean formulas.

- Starting with the maximum strength level and working downward, it derives systems of Boolean equations for each strength level. It can capture the effects of any (fixed) number of signal strengths. These systems of equations express the effects of all paths in the graph but lend themselves to solution methods of polynomial complexity.

- It solves the equations symbolically by Gaussian elimination. Gaussian elimination can exploit the sparse structure of the networks to solve most n node subnetworks with $O(n)$ algebraic operations.

- It represents the set of Boolean formulas as a directed acyclic graph (DAG). This representation naturally allows sharing of common subexpressions. The size of the DAG describing the steady state response of all nodes in a subnetwork is bounded by the number of algebraic operations required during the Gaussian eliminations.

Comparing the analyzer to the inner workings of the circuit-level simulator SPICE [24] lends some insight into the underlying ideas. During transient analysis, SPICE computes the behavior of a nonlinear network at each time point by performing a series of iterations, each of which involves setting up a system of linear equations and solving it by Gaussian elimination. Similarly, our analyzer computes the behavior of a network of non-Boolean switches (due to the different signal strengths and the X states) by performing a series of iterations, each of which involves setting up 3 systems of Boolean equations and solving them by Gaussian elimination. In this respect, both programs apply the powerful mathematical technique of solving a difficult problem over a poorly structured domain by recasting it as a series of problems in a more tractable domain for which efficient, highly developed algorithms exist.

Unlike SPICE, however, the analyzer iterates in a fixed progression over signal strengths rather than until it reaches some convergence criterion. This progression is possible because of the discrete nature of signal strengths. At a given strength level, the analyzer has already computed the effects of stronger signals and can safely ignore weaker ones. Furthermore, rather than being performed on each time step, the analyzer need only compute the behavior once, yielding a set of formulas that are evaluated repeatedly during simulation. Such a symbolic analysis is possible because of the simpler natures of both the circuit elements and the mathematical domain. Whereas SPICE must linearize the network by evaluating complex device models at the current operating points of the circuit elements, the analyzer need only evaluate the effects of the possible paths at each strength level. Furthermore, Boolean formulas are far easier to manipulate and simplify than formulas over real numbers. Thus, while interesting parallels exist between SPICE and symbolic switch-level analysis, many factors contribute to make the latter far more efficient.

A companion paper [25] provides background on the mathematical and algorithmic techniques used in the analysis. This paper gives a detailed formulation of the switch-level model in terms of Boolean algebra. It also describes several extensions to the model, including ways to model circuit faults, degraded logic signals, and charge decay. These extensions demonstrate the power of the basic framework. The analyzer can incorporate new modeling features by modifying the basic systems of equations slightly. Future papers will cover implementation issues, applications, and experimental results.

The remainder of the paper is organized as two major parts. The first, consisting of Sections 2–4, formulates the behavior of a switch-level network as a system of equations in an abstract Boolean algebra. The second part, consisting of Sections 5–7, presents refinements of the technique, examples, and extensions to the switch-level model.

2 The Switch-Level Model

The switch-level model considered here has been described in detail elsewhere [1]. This section gives an overview of the model in terms of a cleaner notation and defines the symbolic analysis problem.

2.1 Network Model

A switch-level network consists of a set of nodes and a set of transistors. A node is classified as either input or storage. An *input* node represents a connection to a signal source external to the chip, supplying either power, ground, clock, or data. A *storage* node, like a capacitor in an electrical network, retains its state in the absence of applied inputs and can share charge with other storage nodes. The voltage on node `n`[1] is represented by its *state* $n \in \{0, 1, X\}$, with 0 and 1 corresponding to low and high voltage levels, respectively, and X corresponding to an indeterminate voltage between low and high indicating an uninitialized network state or an error condition caused by a short circuit or charge sharing.

A storage node has a characteristic *size* from the set $\{1, 2, \ldots, k\}$. This size indicates, in a highly simplified way, the node capacitance relative to that of other nodes with which it may share charge. That is, when a set of storage nodes share charge (due to connections by conducting transistors), only the connected nodes of maximum size determine the outcome. Input nodes are indicated by size $w > k$. The set $\mathcal{N}_s$ contains all nodes of size s, and hence $\mathcal{N}_w$ denotes the set of input nodes.

A transistor has terminals labeled, "gate", "source", and "drain". It acts as a resistive switch connecting the source and drain nodes controlled by the state of the gate node. Transistors act as bidirectional elements with no predetermined direction of information or current flow. A transistor has a *type* indicating the conditions under which it will become conducting. A *d-type* transistor always conducts; an *n-type* conducts when its gate has state 1; while a *p-type* conducts when its gate has state 0. When the gate node of an n-type or p-type transistor has state X, the transistor can range between fully conducting and open circuited. Transistor states 0, 1, and X represent conduction levels nonconducting, fully conducting, and indeterminate, respectively.

Each transistor has a characteristic *strength* from the set $\{k+1, k+2, \ldots, w-1\}$. This strength indicates, in a highly simplified way, the transistor conductance relative to those of other transistors in a ratioed circuit. That is, every path of conducting transistors has a characteristic strength equal to the that of the weakest transistor in the path. When a set

[1]This presentation uses a notation where nodes are named by lower-case letters, e.g., `m`, `n`, their current states are indicated by italicized, lower-case letters, e.g., m, n, and their new states are indicated by italicized, upper-case letters, e.g., M, N.

of paths form from several input nodes to a storage node, only those inputs connected by maximum strength paths determine the new node state. For nodes m and n, the set $\mathcal{T}_s(\mathtt{m},\mathtt{n})$ contains all transistors of strength s having these two nodes as source and drain.

2.2 The Channel Graph

The *channel graph* represents the interconnection structure of a switch-level network. This graph has the storage nodes of the circuit as vertices, and an edge $(\mathtt{m},\mathtt{n})$ for each pair of storage nodes m and n such that $\mathcal{T}_s(\mathtt{m},\mathtt{n}) \neq \emptyset$ for some strength s. It describes the static (independent of transistor state) interconnections between storage nodes formed by the source-drain connections of the transistors. The channel graph is considered either undirected or directed depending on context. When talking about general structural properties of a circuit, an undirected graph simplifies the discussion. On the other hand, symbolic analysis requires a directed graph, because the labels assigned to the edges are direction sensitive.

In general, a channel graph consists of many connected components. Therefore, it defines a partitioning of the switch-level network into a set of *channel-connected subnetworks*, where each subnetwork consists of the set of nodes in a graph component, plus the set of transistors for which these nodes are sources or drains. Note that an input node is not part of any subnetwork, but a transistor for which the node is source (drain) is in the subnetwork of its drain (source) node.

Within a subnetwork, the behavior can be complex and difficult to analyze due to the bidirectional transistors and the many ways state forms in a MOS circuit. The interactions between subnetworks, however, are much more straightforward. Each subnetwork acts as a sequential logic element having as inputs the input nodes connected to transistor sources and drains as well as the gate nodes of the transistors. The subnetwork state is stored as charge on the storage nodes, and the outputs are those nodes that are gate nodes of transistors in other subnetworks. Hence, the overall operation of a switch-level simulator is similar to that of a logic gate simulator—changing values on the subnetwork inputs require updating the state and outputs, and these changing output values in turn affect other subnetworks. The challenge then is to develop formulas representing the behavior of individual subnetworks.

In practice, many subnetworks are small—containing at most 10 transistors. However, we have encountered subnetworks with over 5000 transistors (essentially the entire data path of a 16-bit microprocessor [26]), and hence the analysis of each subnetwork must be as efficient as possible.

2.3 Steady State Response

The steady state response function describes the behavior of a subnetwork. Informally, this function can be explained as follows. For a given set of connected input node and initial storage node states, the transistors are set according to their gate node states. The transistors in the 1 and X states create (potentially) conducting paths from input nodes to storage nodes and between pairs of storage nodes, causing the storage nodes to attain new voltage levels. The steady state response for a node equals the state (0, 1, or X) this node would attain if the transistors were held fixed long enough for the nodes to stabilize. When nodes or transistors in the X state are present, the steady state response on a node equals 0 or 1 only when it would attain this unique state regardless of the voltages and conductances of these nodes and transistors. Otherwise the steady state response equals X.

The steady state response for a subnetwork is defined formally in terms of the paths between nodes formed by the conducting transistors. This approach unifies the variety of different ways logic values form in MOS circuits including: stored charge, charge sharing, and both ratioed and complementary logic. For a given set of transistor states, transistors in the 1 and X state form connections between their source and drain nodes. A *rooted path* p is a directed path originating at node $Root(p)$, terminating at node $Dest(p)$ and consisting of a (possibly empty) set of transistors $Trans(p)$. The *strength* of path p, denoted $|p|$ is defined as

$$|p| \;\; = \;\; \min \left[Size[Root(p)], \;\; \min_{t \in Trans(p)} Strength(t) \right]$$

A rooted path represents a source of charge from its root to its destination with driving ability indicated by its strength. Rooted paths can be classified into three types according to their strength. A path with strength $1 \leq |p| \leq k$ represents a source of stored charge from a storage node with an approximate capacitance determined by the size of this node. Note that the stored charge initially on the node is represented by a path with root and destination equal to the node and with no transistors. A path with strength $k < |p| < w$ represents a source of current from an input node with an approximate conductance determined by the strength of the weakest transistor in the path. A path with strength $|p| = w$ must contain no edges and have an input node as both root and destination. Such a path represents the external current supplied to the input node. The overall ranking of path strengths reflects the fact that a connection from an input node can override any stored charge, while a direct connection to an input can override any resistive connection from some other input.

The steady state response of a node depends only on the paths to the node that are not "blocked". A *definite* path is defined as a rooted path p such that no transistor in $Trans(p)$ is in the X state. A path p is said to be *blocked* if for some initial segment p' of p (i.e. $Root(p') = Root(p)$ and $Trans(p') \subseteq Trans(p)$) and for some *definite* path q, $Dest(p') = Dest(q)$ and $|p'| < |q|$. Intuitively, a path is blocked if the source of charge it represents would be overridden by a stronger source at some intermediate node. Define the path relation $\mathcal{P}$ between pairs of nodes as $\mathtt{m} \, \mathcal{P} \, \mathtt{n}$ when there is an unblocked path p with $Root(p) = \mathtt{m}$ and $Dest(p) = \mathtt{n}$. Then the steady state response on node $\mathtt{n}$, denoted N, is given by the equation

$$N \;\; = \;\; \mathrm{lub}\,\{m | \mathtt{m} \, \mathcal{P} \, \mathtt{n}\}, \tag{1}$$

where "lub" represents the least upper bound over the ordering $0 < X$ and $1 < X$. In other words, if all unblocked sources of charge to a node drive it to 0 (or to 1), then the steady state response equals 0 (or 1). Otherwise, if the node is driven by conflicting sources or by sources of unknown value, the steady state response equals X. It can be shown that this characterization of the steady state response provides an accurate modeling of the effects of unknown states as well as several important mathematical properties [1].

2.4 State Encoding

To cast the switch-level model in terms of Boolean operations, a state value $y \in \{0, 1, X\}$ is encoded as two Boolean values $y.1, y.0 \in \{0, 1\}$ as follows

y	$y.1$	$y.0$
1	1	0
0	0	1
X	1	1

Formally, .1 and .0 are operators, expressed in postfix notation, mapping elements of $\{0, 1, X\}$ to elements of $\{0, 1\}$. The combination $y.1 = y.0 = 0$ does not represent a valid state. It can be considered a "don't care" combination in the derivation. With this encoding, if y is the least upper bound of a set A consisting of elements $a \in \{0, 1, X\}$, then

$$y.1 = \bigvee_{a \in A} a.1 \tag{2}$$

$$y.0 = \bigvee_{a \in A} a.0, \tag{3}$$

where $\bigvee$ denotes the Boolean sum of a set of elements.

With this Boolean encoding of state values, the symbolic analysis problem can be defined as follows. For each node `n`, introduce Boolean variables $n.1$ and $n.0$ to represent the encoded value of the initial node state. Of course, when the node is known to have a fixed state (e.g., power or ground), its state can be encoded by constants rather than variables. For each node `n`, we are to derive Boolean formulas, denoted $N.1$ and $N.0$, for the encoded steady state response in terms of the node state variables. The encoding of node states makes it possible to express the three-valued circuit behavior using conventional Boolean algebra. This greatly simplifies the algebraic manipulation portion of the symbolic analyzer, at the cost of requiring a pair of formulas to describe each node.

In terms of this encoding, Equations 2 and 3 can be applied to Equation 1 to give:

$$N.1 = \bigvee_{\mathtt{m}\mathcal{P}\mathtt{n}} m.1 \tag{4}$$

$$N.0 = \bigvee_{\mathtt{m}\mathcal{P}\mathtt{n}} m.0. \tag{5}$$

3 Mathematical and Algorithmic Background

This section briefly summarizes the mathematical notation, results, and algorithms developed in the companion paper.

3.1 Symbolic Algebra

A Boolean formula describes a function mapping each possible combination of values for the set of p variables to 0 or 1. Mathematically, symbolic analysis can be viewed as manipulating elements of the algebra $\langle \mathcal{B}, \wedge, \vee, \neg, \mathbf{0}, \mathbf{1} \rangle$, where

$$\mathcal{B} \quad = \quad \left\{ f \colon \{0,1\}^p \rightarrow \{0,1\} \right\}.$$

The operations $\wedge$, $\vee$, and $\neg$ denote Boolean AND, OR, and NOT, respectively, applied to functions. The distinguished elements $\mathbf{0}$ and $\mathbf{1}$ represent the constant functions that yield 0 and 1, respectively, for all argument values. This process of *abstracting* from a primitive domain to one of functions, while maintaining the algebraic properties, forms the basis of symbolic analysis.

The Boolean product of the elements in a set A is denoted $\bigwedge_{a \in A} a$. The product of an empty set is defined to equal $\mathbf{1}$. Similarly, the Boolean sum of the elements in a set A is denoted $\bigvee_{a \in A} a$. The sum of an empty set is defined to equal $\mathbf{0}$.

3.2 Systems of Boolean Equations

Systems of Boolean equations provide a mathematical formalism for networks of switches much as do systems of linear equations for networks of resistors. However, to emphasize the sparse nature of the networks, labeled graphs are preferred to a matrix notation.

A system of Boolean equations is represented by an edge and vertex labeling on a directed graph (V, E), where a labeling indicates of an assignment of elements of $\mathcal{B}$ to every edge or vertex. The system $[A, b]$ consists of an edge label $A(u, v) \in \mathcal{B}$ for each $(u, v) \in E$ and a vertex label $b(v) \in \mathcal{B}$ for each $v \in V$. Vertex labeling x *satisfies* the system $[A, b]$ when

$$x(v) \quad = \quad b(v) \vee \bigvee_{(u,v) \in E} [x(u) \wedge A(u, v)]$$

for every $v \in V$. In general, many labelings may satisfy a system, but by defining an appropriate partial ordering of the elements of $\mathcal{B}$, every system can be shown to have a unique minimum satisfying labeling. This labeling is termed the system *solution*.

The solution of a Boolean system describes the conditions under which conducting paths will form in a switch network. More precisely, $P_{u,v}$ is defined as the set of all paths from vertex u to v in the graph. For solution x of the system $[A, b]$:

$$x(v) \quad = \quad \bigvee_{u \in V} \; \bigvee_{p \in P_{u,v}} \left[b(u) \wedge \bigwedge_{(s,t) \in p} A(s, t) \right]$$

for all vertices v. In other words, if the "value" of a path is defined as the Boolean product of the initial vertex label and all edge labels, then $x(v)$ equals the Boolean sum of the values of all paths terminating at v.

A *dual* system of Boolean equations is similar to a normal system, but with the roles of $\wedge$ and $\vee$ interchanged. Labeling x satisfies the dual system $[A, b]^D$ when

$$x(v) \quad = \quad b(v) \wedge \bigwedge_{(u,v) \in E} [x(u) \vee A(u, v)]$$

for every $v \in V$. Dual systems express conditions under which conducting paths are absent in a switch network. More precisely, let $\overline{A}$ denote the edge labeling with each element equal to the Boolean complement of the corresponding element of A, and similarly for $\overline{b}$. For solution x of the dual system $[\overline{A}, \overline{b}]^D$:

$$x(v) \quad = \quad \neg \bigvee_{u \in V} \ \bigvee_{p \in P_{u,v}} \left[b(u) \wedge \bigwedge_{(s,t) \in p} A(s,t) \right]$$

Gaussian elimination can solve a system of Boolean equations (either normal or dual), with operations $\wedge$ and $\vee$ replacing the real arithmetic used when solving linear systems. Most channel graphs fall into a class called General Series-Parallel (GSP). This class includes both conventional series-parallel graphs as well as ones containing acyclic branches. Gaussian elimination requires at most $12n$ algebraic operations to solve a Boolean system defined over an n vertex GSP graph.

3.3 Boolean Formula Representation

A directed acyclic graph (DAG), with leaves denoting variables and constants and with nodes denoting Boolean operations, can represent a set of Boolean formulas. The symbolic solution of a Boolean system generates a DAG with each formula indicated by a pointer to some DAG node. The symbolic manipulator applies a Boolean operation to two formulas by creating a new node with branches to the nodes representing the arguments. By this means, the total size of the Boolean description generated is bounded by the number of algebraic operations performed during Gaussian elimination. The manipulator applies graph transformation rules corresponding to the laws of Boolean algebra to simplify the formulas, thereby reducing the DAG size.

A given Boolean function has many different DAG representations. The exact formula structure generated during Gaussian elimination depends on the order in which vertices are eliminated. Hence, the result is neither unique nor of minimum size. Using only these "weak" symbolic manipulation algorithms, however, avoids trying to solve any NP-hard problems.

4 Boolean Representation of the Steady State Response

This section formulates the steady state response of a node in terms of Boolean operations as well as relations and predicates between the nodes. It then develops systems of Boolean equations to describe the steady state response symbolically.

4.1 Strength Encoding

The analyzer accounts for the effects of different strength signals by starting at the maximum strength and working downward, each time adding in the effects from paths of the next lower strength. This approach captures signal strength effects in the *structure* of the equations to

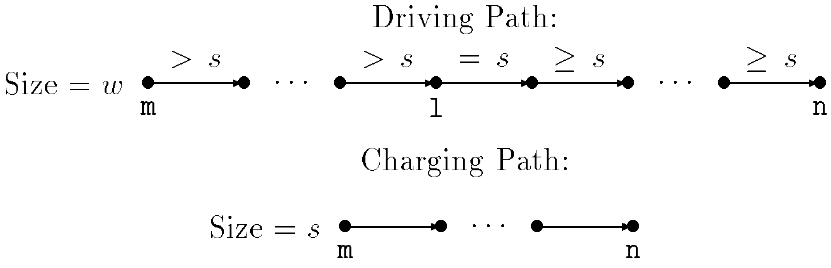

Figure 1: **General Form of a Strength s Path.** For $s > k$, the path originates at an input node, passes through a strength s transistor, and contains no weaker transistors. For $s \leq k$, the path originates at a storage node of size s.

be solved. In contrast, MOSSIM II and most other switch-level simulators encode strength effects in the *algebra* in which the equations are expressed. This structural approach makes it possible to express the behavior in terms of Boolean algebra. It has the disadvantage that the number of equations to be solved is proportional to the total number of signal strengths w, whereas the algebraic approach can use algorithms with complexity essentially independent of w. This does not compromise the efficiency significantly, however, because few MOS circuits require more than 6 signal strengths to characterize their behavior (2 storage node sizes, 3 transistor strengths, and 1 input node size.)

For a given set of transistor states and for signal strength s, the path relation $\mathcal{P}_s$ is defined as $\mathtt{m}\ \mathcal{P}_s\ \mathtt{n}$ when there exists an unblocked path of strength greater than or equal to s from `m` to `n`. If we define $N.1_s$ as

$$N.1_s \quad = \quad \bigvee_{\mathtt{m}\mathcal{P}_s\mathtt{n}} m.1 \tag{6}$$

and $N.0_s$ as

$$N.0_s \quad = \quad \bigvee_{\mathtt{m}\mathcal{P}_s\mathtt{n}} m.0 \tag{7}$$

then $N.1_s$ (respectively, $N.0_s$) describes the conditions under which node `n` will be the destination of an unblocked path of strength s or greater originating at a node `m` with $m = 1$ or X, (resp., 0 or X.) By this definition, $N.1 = N.1_1$ and $N.0 = N.0_1$ for node `n`.

Consider the general form of an unblocked path from node `m` to node `n` having strength greater than or equal to s. It can be an unblocked path of strength greater than s, in which case $\mathtt{m}\ \mathcal{P}_{s+1}\ \mathtt{n}$. Otherwise, the path must have strength s, the possible forms of which are illustrated in Figure 1. For a driving path, node `m` must be an input node, and the path must consist of a (possibly empty) sequence of transistors of strength greater than s to some node `l`, followed by a transistor of strength s, followed by a (possibly empty) sequence of transistors of strength greater than or equal to s to `n`. The portion from `m` to `l` cannot be blocked, and hence $\mathtt{m}\ \mathcal{P}_{s+1}\ \mathtt{l}$. Furthermore, no node in the portion from `l` to `n`, except `l`, can be the destination of a definite path of strength greater than s. For a charging path, node `m` must be a storage node of size s, and the path must consist of a sequence of transistors from `m` to `n` such that no node is the destination of a definite path of strength greater than s.

These conditions can be incorporated into a formal definition of $\mathcal{P}_s$ by introducing additional predicates and relations. The conditions expressed by these conditions and relations can then be expressed symbolically as the solutions to systems of Boolean equations. For each node $\mathtt{n}$ define the predicate $\mathcal{C}_s(\mathtt{n})$ as holding when $\mathtt{n}$ is *not* the destination of any definite path of strength greater than or equal to s. This predicate expresses the condition that the node is *clear*, i.e., not blocked, for signals of strength $s-1$. Define the relation $\mathcal{Q}_s$ as $\mathtt{m}\ \mathcal{Q}_s\ \mathtt{n}$ when the following conditions hold:

- There is a path p in the network with $Root(p) = \mathtt{m}$ and $Dest(p) = \mathtt{n}$ consisting only of transistors with state 1 or X and strength greater than or equal to s.
- $\mathcal{C}_{s+1}(\mathtt{l})$ holds for every node $\mathtt{l}$ in p other than $\mathtt{m}$.

The relation $\mathcal{P}_s$ can then be expressed as $\mathcal{P}_w = \{(\mathtt{n},\mathtt{n}) | \mathtt{n} \in \mathcal{N}_w\}$, and for $s < w$:

$$\begin{aligned} \mathcal{P}_s &= \mathcal{P}_{s+1} \cup \left\{ (\mathtt{m},\mathtt{n}) \middle| \exists \mathtt{l},\ \mathtt{m}\ \mathcal{P}_{s+1}\ \mathtt{l} \text{ and } \mathtt{l}\ \mathcal{Q}_s\ \mathtt{n} \right\} \\ &\quad \cup \left\{ (\mathtt{m},\mathtt{n}) \middle| \mathtt{m}\ \mathcal{Q}_s\ \mathtt{n},\ \mathcal{C}_{s+1}(\mathtt{m}),\ \text{and } \mathtt{m} \in \mathcal{N}_s \right\} \end{aligned}$$

The three terms in this equation represent the three classes of unblocked paths having strength greater than or equal to s discussed above.

Substituting for the definition of $N.1_s$ gives

$$N.1_s \quad = \quad \bigvee_{\mathtt{m}\mathcal{P}_{s+1}\mathtt{n}} m.1 \ \vee \ \bigvee_{\mathtt{m}\mathcal{P}_{s+1}\mathtt{l}} \ \bigvee_{\mathtt{l}\mathcal{Q}_s\mathtt{n}} m.1 \ \vee \ \bigvee_{\mathtt{m}\mathcal{Q}_s\mathtt{n}} \mathcal{C}_{s+1}(\mathtt{m}) \wedge (\mathtt{m} \in \mathcal{N}_s) \wedge m.1$$

By Equation 6 the first term in this equation equals $N.1_{s+1}$. The second term can be transformed by reversing the ordering of the summation, applying Equation 6 inside the summation, and changing the summation variable as follows:

$$\bigvee_{\mathtt{m}\mathcal{P}_{s+1}\mathtt{l}} \ \bigvee_{\mathtt{l}\mathcal{Q}_s\mathtt{n}} m.1 \quad = \quad \bigvee_{\mathtt{l}\mathcal{Q}_s\mathtt{n}} \left(\bigvee_{\mathtt{m}\mathcal{P}_{s+1}\mathtt{l}} m.1 \right) \quad = \quad \bigvee_{\mathtt{l}\mathcal{Q}_s\mathtt{n}} L.1_{s+1} \quad = \quad \bigvee_{\mathtt{m}\mathcal{Q}_s\mathtt{n}} M.1_{s+1}.$$

The equation for $N.1_s$ then becomes

$$N.1_s \quad = \quad N.1_{s+1} \ \vee \ \bigvee_{\mathtt{m}\mathcal{Q}_s\mathtt{n}} \Big(M.1_{s+1} \vee \big[\mathcal{C}_{s+1}(\mathtt{m}) \wedge (\mathtt{m} \in \mathcal{N}_s) \wedge m.1 \big] \Big). \tag{8}$$

By similar reasoning, $N.0_s$ can be written as

$$N.0_s \quad = \quad N.0_{s+1} \ \vee \ \bigvee_{\mathtt{m}\mathcal{Q}_s\mathtt{n}} \Big(M.0_{s+1} \vee \big[\mathcal{C}_{s+1}(\mathtt{m}) \wedge (\mathtt{m} \in \mathcal{N}_s) \wedge m.0 \big] \Big). \tag{9}$$

Thus the steady state response at strength s is expressed in terms of the response at strength $s+1$, the relation $\mathcal{Q}_s$ and the predicate $\mathcal{C}_{s+1}$. Equations for both $\mathcal{Q}_s$ and $\mathcal{C}_{s+1}$ can be formulated as systems of Boolean equations, thereby formulating the steady state response in Boolean terms. Interestingly, this derivation yields a result similar to one derived by Byrd, Hachtel, and Lightner [27] based on an "order of magnitude" linear network model.

They show that at each strength level, the effect of all stronger signals to a node can be represented by a single voltage source with voltage corresponding to the net signal value, while all weaker signals can be ignored. Terms of the form $M.1_{s+1}$ and $M.0_{s+1}$ in Equations 8 and 9 are analogous to a source at node $\mathtt{m}$ representing the net effect of the stronger signals at this node. Furthermore, these equations contain no terms representing signals of strength less than s.

4.2 Symbolic Formulation

With this background, we are ready to formulate the steady state response in terms of systems of Boolean equations. For a transistor t, the formulas $\mathit{indefinite}(t)$ and $\mathit{potential}(t)$ indicate whether the transistor is not definitely conducting (in state 0 or X) or potentially conducting (in state 1 or X) depending on the state of its gate node $\mathtt{n}$ as follows:

type	$\mathit{indefinite}(t)$	$\mathit{potential}(t)$
n-type	$n.0$	$n.1$
p-type	$n.1$	$n.0$
d-type	$\mathbf{0}$	$\mathbf{1}$

Let (V, E) be a directed graph corresponding to a single component of the channel graph. For each strength level s, such that $w > s \geq 1$, the analyzer sets up and solves three systems of Boolean equations (two normal and one dual) for different labelings of this graph. The solutions yield formulas for $N.1_s$, $N.0_s$, and $\mathcal{C}_{s+1}(\mathtt{n})$ for each storage node $\mathtt{n}$ in the subnetwork. In each case, the edge labeling describes the transistor connections between pairs of storage nodes, while the vertex labeling describes a combination of the initial value on the storage node plus those on input nodes connected by single transistors. This approach takes advantage of the fact that any network path passing through an input node must be blocked, and hence transistors connected to input nodes act as unidirectional switches.

Starting with $N.1_w = \mathbf{0}$ for any storage node $\mathtt{n}$, the analyzer computes formulas for $N.1_s$, $w > s \geq 1$ by solving the system $[\mathit{Conduct}_s, \mathit{init}1_s]$. This system of equations is based on Equation 8. The edge labeling $\mathit{Conduct}_s$ expresses the conditions under which a sequence of transistors satisfies the conditions of the relation $\mathcal{Q}_s$:

$$\mathit{Conduct}_s(\mathtt{m},\mathtt{n}) \quad = \quad N.c_{s+1} \wedge \left[\mathit{Conduct}_{s+1}(\mathtt{m},\mathtt{n}) \vee \bigvee_{t \in \mathcal{T}_s(\mathtt{m},\mathtt{n})} \mathit{potential}(t)\right], \tag{10}$$

where $\mathit{Conduct}_w(\mathtt{m},\mathtt{n}) = \mathbf{0}$. The term $N.c_{s+1}$ is a formula that symbolically expresses the predicate $\mathcal{C}_{s+1}(\mathtt{n})$, as will be defined shortly. Note the asymmetry in the above edge labeling, where in general $\mathit{Conduct}_s(\mathtt{m},\mathtt{n}) \neq \mathit{Conduct}_s(\mathtt{n},\mathtt{m})$. It arises from the requirement that $\mathcal{C}_{s+1}(\mathtt{l})$ must hold for all nodes $\mathtt{l}$ in the path *other* than the first one. By forming an edge label equal to the Boolean product of the conduction condition for the corresponding transistors and the clear condition for the edge destination, any path formed by these edges must be clear at all but the first node. The vertex labeling $\mathit{init}1_s$ combines terms inside the summation of Equation 8 for a storage node and for connected input nodes:

$$init1_s(\mathtt{n}) \quad = \quad \begin{cases} N.1_{s+1} \;\vee\; \left(N.c_{s+1} \wedge \bigvee_{\mathtt{m}\in\mathcal{N}_w} \bigvee_{t\in\mathcal{T}_s(\mathtt{m},\mathtt{n})} [potential(t) \wedge m.1]\right) & s > k \\ N.1_{s+1} \;\vee\; (N.c_{s+1} \wedge n.1) & \mathtt{n} \in \mathcal{N}_s \\ N.1_{s+1} & \text{else} \end{cases} \tag{11}$$

Starting with $N.0_w = \mathbf{0}$, the analyzer computes formulas for $N.0_s$, $w > s \geq 1$ by solving the system $[Conduct_s, init0_s]$. This system is based on Equation 9. The edge labeling $Conduct_s$ is the same as before (Equation 10.) Vertex labeling $init0_s$ is analogous to $init1_s$:

$$init0_s(\mathtt{n}) \quad = \quad \begin{cases} N.0_{s+1} \;\vee\; \left(N.c_{s+1} \wedge \bigvee_{\mathtt{m}\in\mathcal{N}_w} \bigvee_{t\in\mathcal{T}_s(\mathtt{m},\mathtt{n})} [potential(t) \wedge m.0]\right) & s > k \\ N.0_{s+1} \;\vee\; (N.c_{s+1} \wedge n.0) & \mathtt{n} \in \mathcal{N}_s \\ N.0_{s+1} & \text{else} \end{cases} \tag{12}$$

Finally, starting with $N.c_w = \mathbf{1}$ for any storage node $\mathtt{n}$, the analyzer computes formulas for $N.c_s$, $w > s > 1$ by solving the dual system $[Indef_s, initc_s]^D$. This formula symbolically encodes the predicate $\mathcal{C}_s(\mathtt{n})$. The computation is formulated as a dual system to express the absence of blocking paths. The edge labeling $Indef_s$ describes the conditions under which two storage nodes are *not* connected by a transistor in the 1 state of strength greater than or equal to s:

$$Indef_s(\mathtt{m},\mathtt{n}) \quad = \quad Indef_{s+1}(\mathtt{m},\mathtt{n}) \wedge \bigwedge_{t\in\mathcal{T}_s(\mathtt{m},\mathtt{n})} indefinite(t), \tag{13}$$

where $Indef_w(\mathtt{m},\mathtt{n}) = \mathbf{1}$. The vertex labeling $initc_s$ indicates the conditions under which a storage node neither has size s, nor is the destination of a definite path of strength greater than s, nor is connected to an input node by a transistor with state 1 and strength s:

$$initc_s(\mathtt{n}) \quad = \quad \begin{cases} N.c_{s+1} \wedge \bigwedge_{\mathtt{m}\in\mathcal{N}_w} \bigwedge_{t\in\mathcal{T}_s(\mathtt{m},\mathtt{n})} indefinite(t) & s > k \\ \mathbf{0} & \mathtt{n} \in \mathcal{N}_s \\ N.c_{s+1} & \text{else} \end{cases} \tag{14}$$

To summarize, the computation of the steady state response formulas starts with $s = w-1$ and works downward to $s = 1$. At each strength level the analyzer sets up and solves equations to compute $N.1_s$ and $N.0_s$ for every node. It then sets up and solves a dual system to compute $N.c_s$ for every node for use at the next lower strength level. The desired results for node $\mathtt{n}$ equal $N.1_1$ and $N.0_1$, respectively. Although the above presentation used names subscripted by s to represent the terms at different strength levels, the implementation need only retain the terms for the current strength as it iterates.

5 Refinements

The analyzer as described so far achieves good asymptotic performance in terms of the size of the formulas generated. However, the performance can be further improved by reducing the constant of proportionality, by generating a hierarchical description, or by maximizing the potential concurrency of the evaluation.

5.1 Nonessential Node Elimination

Until now, the presentation has assumed that the analyzer must compute the steady state response for every node in a subnetwork, as is done by most switch-level simulators. However, some nodes serve only as interconnection points in a circuit—they neither control any transistors nor form part of the circuit memory. For example, all intermediate nodes in the pullup and pulldown networks of nMOS and CMOS logic gates serve only as interconnections. For modeling circuit behavior, a program such as a simulator need only keep track of the states of "essential" nodes, i.e., those that can either directly or indirectly affect the value of a subnetwork output.

Once the analyzer has generated the DAG for a subnetwork, a postprocessor can prune it to include only those parts required to compute the states of essential nodes as follows. The postprocessor starts by marking the DAG nodes representing all formulas $N.1$ and $N.0$ for which `n` is a primary output of the circuit or is the gate of an n-type or p-type transistor. It then traces down the DAG and marks their descendants. If it encounters a leaf representing variable $m.1$ (respectively, $m.0$), and the DAG node representing the formula $M.1$ (respectively, $M.0$) has not been marked, then it marks this node and traces the descendants. This process continues until it can reach no further DAG nodes. The pruned DAG consists of those parts that have been marked.

Note that the degree of pruning depends on the degree to which the original formulas have been simplified, because simplification will typically reduce the number of variables occurring in a formula. Therefore, Boolean simplification tends to have a multiplicative effect—greater simplification reduces the size of the original DAG and also increases the amount by which it can be pruned.

5.2 Hierarchical Analysis

The presentation has also assumed that the analyzer must extract the function of every subnetwork in a circuit. However, most VLSI circuits contain repeated structures, and therefore many isomorphic subnetworks. A more efficient method would analyze only unique subnetworks. It would then produce a hierarchical representation in which each subnetwork instance references the appropriate Boolean description with its own set of node parameters. This hierarchical analysis would require less time and produce a more compact description. Such an approach, however, requires a method to recognize isomorphic subnetworks.

A circuit described hierarchically already has much of the commonality represented explicitly. Unfortunately, the hierarchical partitioning of the circuit will not, in general, conform to the partitioning required for symbolic analysis. That is, the circuit elements in a single channel-connected subnetwork may be declared in several components of the hierarchical description. To exploit this hierarchy, the analyzer must first modify the circuit description to respect subnetwork boundaries. It can do this by "pulling" all node and transistor specifications for each subnetwork up the hierarchy into the least common ancestor of the components in which they originally occurred.

Alternatively, the analyzer can extract common subnetworks by applying graph isomorphism techniques. Although efficient and reliable algorithms for general graph isomorphism have not yet been developed, heuristic methods developed in the context of interconnect verification have proved very successful [32, 33]. Furthermore, if the analyzer fails to recognize some isomorphisms, the output will not be as compact, but the results will still be valid.

5.3 Maximizing Potential Parallelism

In the near future, preprocessors for switch-level networks will routinely generate code for computers that support high degrees of parallelism. For example, the YSE can have up to 256 processors operating simultaneously and communicating through a cross-bar switch. Under such conditions, it is more important for the preprocessor to reduce sequential constraints imposed by data dependencies rather than to minimize the formula size. Reducing data dependencies also simplifies scheduling on highly pipelined processors. As is mentioned in the companion paper, pivots can be chosen for Gaussian elimination such that the analysis of an `n` node, general series-parallel network yields a set of formulas with $O(n)$ total operations and maximum depth $O(\log\, n)$. Thus, given sufficient parallel resources, the steady state response for a subnetwork could be computed in sublinear time. In contrast, the algorithms used by MOSSIM II and all other switch-level simulators cannot achieve sublinear performance regardless of the processing capabilities. For example, they would be effectively limited to sequential execution when propagating a signal down a long chain of transistors. Gaussian elimination removes this constraint by collecting information about the entire chain and then redistributing the results, each time through expression trees of logarithmic depth. Thus a preprocessor based on Gaussian elimination becomes especially attractive for highly parallel systems.

6 Examples

This section highlights some characteristics of the analyzer by evaluating how the analysis would proceed for several general MOS implementations of logic gates, and by executing the algorithm on a small CMOS circuit. For circuits containing more than a handful of transistors, it becomes impractical to trace the execution steps in detail, and the resulting formulas are too large to examine manually. In studying these examples, the reader must keep in mind that the true strength of the analyzer lies in its ability to handle much larger circuits.

The presentation expresses performance by a parameter α, defined as the the total number of binary Boolean operations in the formulas divided by the number of transistors in the network being analyzed. Lower values of α indicate a more efficient analysis. Although this parameter only measures the size of the analyzer output, it also provides a reasonable indication of the time required for execution. A worst case analysis shows that α cannot exceed 240 for circuits with at most 6 signal strengths where all subnetworks have general series-parallel channel graphs [25]. This analysis, however, is far too pessimistic. Experiments on actual circuits indicate a typical range of 2 to 10.

6.1 General Logic Gates

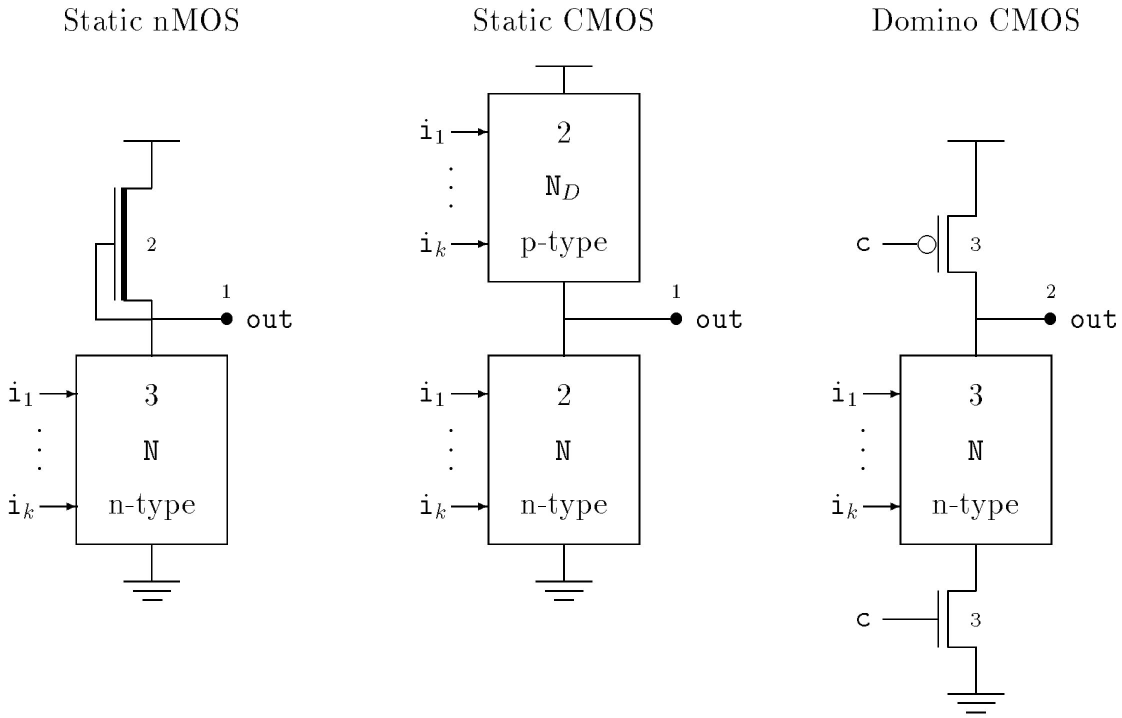

Figure 2: **Switch-level Representations of Logic Gates.** The boxes indicate networks of transistors labeled by strength and type. Transistors are labeled by their strength, and storage nodes by their size.

Figure 2 illustrates the switch-level representations of three classes of MOS logic gates. In this figure the network N represents a pulldown network of n-type transistors. When viewed as a two terminal network of switches with control variables $a_1, \ldots, a_k$, the conditions under which a path forms across the terminals is given by its *transmission function* $T(a_1, \ldots, a_k)$. Similarly the network $\mathtt{N}_D$ represents a pullup network of p-type transistors. When viewed as a network of *positive* switches this network has a transmission function $T_D(a_1, \ldots, a_k)$ equal to the *dual* of T. That is, these two functions are related as

$$T(a_1, \ldots, a_k) = \neg T_D(\neg a_1, \ldots, \neg a_k).$$

Note also that T is the dual of T_D. In most cases, `N` is a series-parallel network with $\mathtt{N}_D$ its dual. That is, parallel connections in `N` correspond to series connections in $\mathtt{N}_D$, and *vice-versa*. However, these conditions are not mandatory—network duality is a sufficient, but not necessary, condition for functional duality. The formula obtained by solving a system of equations representing network `N` is equivalent to that obtained by solving a dual system of equations representing network $\mathtt{N}_D$, and *vice-versa*.

A static nMOS gate consists of a pulldown network of n-type transistors connecting the output to ground, and a weaker, depletion mode transistor connecting the output to power. The storage node sizes make no difference in the result and are set to 1 for the example. The analysis of the steady state response at node `out` proceeds as follows:

$$\begin{aligned}
Out.1_3 &= \mathbf{0} \\
Out.0_3 &= T(i_1.1, \ldots, i_k.1) \\
Out.c_3 &= T_D(i_1.0, \ldots, i_k.0) \\[1em]
Out.1_2 &= Out.c_3 \wedge \mathbf{1} \qquad = T_D(i_1.0, \ldots, i_k.0) \\
Out.0_2 &= T(i_1.1, \ldots, i_k.1) \\
Out.c_2 &= \mathbf{0} \\[1em]
Out.1_1 &= T_D(i_1.0, \ldots, i_k.0) \\
Out.0_1 &= T(i_1.1, \ldots, i_k.1)
\end{aligned}$$

giving $Out.1 = T_D(i_1.0, \ldots, i_k.0)$ and $Out.0 = T(i_1.1, \ldots, i_k.1)$. That is, the formula for $Out.0$ would express the function T applied to variables $i_1.1, \ldots, i_k.1$, while the formula for $Out.1$, arising from the dual analysis of network `N`, would express the function T_D applied to $i_1.0, \ldots, i_k.0$.

For example, a NAND gate with inputs `a` and `b` would have steady state response functions:

$$\begin{aligned}
Out.1 &= a.0 \vee b.0 \\
Out.0 &= a.1 \wedge b.1
\end{aligned}$$

Evaluating these formulas for all possible values of the variables yields the functionality shown in Table 6.1. As can be seen, these formulas capture the conventional ternary extension of the NAND function. In general, the pairs of formulas for all nMOS logic gates express a ternary behavior equal to the ternary extension of the corresponding gate function [28].

Node `out` is always the destination of a definite path of strength 2, and hence the stored charge of the nodes within `N` cannot affect its steady state response. Eliminating nonessential

a	$a.1$	$a.0$	b	$b.1$	$b.0$	$Out.1$	$Out.0$	Out
0	0	1	0	0	1	1	0	1
0	0	1	1	1	0	1	0	1
0	0	1	X	1	1	1	0	1
1	1	0	0	0	1	1	0	1
1	1	0	1	1	0	0	1	0
1	1	0	X	1	1	1	1	X
X	1	1	0	0	1	1	0	1
X	1	1	1	1	0	1	1	X
X	1	1	X	1	1	1	1	X

Table 1: **Three-valued Behavior of** NAND **Gate**. The formulas generated by the analyzer predict the same behavior as conventional ternary logic.

variables would then reduce the set of formulas to those expressing the steady state response of `out`. For the case of a series-parallel network, and an analysis by Gaussian elimination with node `out` eliminated last, the size of the formula generated will equal the number of transistors minus 1. This gives a performance measure α slightly less than 2. Even for networks that are not strictly series-parallel, such as ones containing bridges, α will not significantly exceed 2.

A static CMOS gate consists of a pulldown network of n-type transistors connecting the output to ground and a pullup network of p-type transistors connecting the output to power, where the two networks have dual transmission functions. Neither the transistor strengths nor the storage node sizes affect the gate function. For the example they are set to 2 and 1, respectively. The analysis of the steady state response at node `out` proceeds as follows:

$$\begin{aligned}
Out.1_2 &= T_D(i_1.0, \ldots, i_k.0) \\
Out.0_2 &= T(i_1.1, \ldots, i_k.1) \\
Out.c_2 &= T(i_1.1, \ldots, i_k.1) \wedge T_D(i_1.0, \ldots, i_k.0)
\end{aligned}$$

$$\begin{aligned}
Out.1_1 &= T_D(i_1.0, \ldots, i_k.0) \vee [x \wedge Out.c_2] &&= T_D(i_1.0, \ldots, i_k.0) \\
Out.0_1 &= T(i_1.1, \ldots, i_k.1) \vee [y \wedge Out.c_2] &&= T(i_1.1, \ldots, i_k.1)
\end{aligned}$$

where the terms x and y express the effects of the initial stored charge on nodes `out` and those internal to `N` and $\mathtt{N}_D$. These terms are eliminated by absorption and hence are not shown in detail. The steady state response is therefore identical to that obtained for an equivalent nMOS gate even when some inputs equal X.

Unlike nMOS gates, there may be no definite driving path to `out`, and hence the paths representing sources of stored charge to `out` may not be blocked. However, in all such cases the gate output will equal X, and hence these paths have no effect. Unfortunately, the analyzer may not recognize the possible absorption of terms representing sources of stored charge by those representing driving paths. To do so, it must recognize that the formulas generated during the normal analysis of `N` and $\mathtt{N}_D$ are equivalent to those generated during the dual analysis of $\mathtt{N}_D$ and N, respectively. Using Gaussian elimination where node `out` is eliminated last, and the formula manipulation techniques described in the companion paper, these equivalences will be recognized for the most common case of N being series-parallel and $\mathtt{N}_D$ its dual. If these absorption conditions are recognized, then nonessential variable elimination will reduce the set of formulas to those representing the steady state response of `out`. Therefore, for series-parallel networks, the analyzer has a performance with α slightly less than 1. On the other hand, it will not do as well for networks that are not series-parallel, nor where N and $\mathtt{N}_D$ are not dual networks. Such cases are sufficiently rare to have little impact on the overall performance.

A domino CMOS gate [29, 30] consists of a p-type precharge transistor, a pulldown network of n-type transistors, and an n-type discharge transistor that can connect the gate output to ground through the pulldown network. Both the precharge and the discharge transistors are gated by a common clock `c`. Transistor strengths do not affect the gate function and are set to 3 for the example. However, when connected in domino fashion, the output node must have greater capacitance than the internal nodes of N, because it may share charge with them. Therefore node `out` has size 2 and the nodes internal to N have size 1. The analysis of the steady state response at node `out` would proceed as follows:

$$\begin{aligned}
Out.1_3 &= c.0 \\
Out.0_3 &= T(i_1.1, \ldots, i_k.1) \wedge c.1 \\
Out.c_3 &= [T_D(i_1.0, \ldots, i_k.0) \vee c.0] \wedge c.1 \\
&= [T_D(i_1.0, \ldots, i_k.0) \wedge c.1] \vee [c.1 \wedge c.0] \\
\\
Out.1_2 &= c.0 \vee [T_D(i_1.0, \ldots, i_k.0) \wedge c.1 \wedge out.1] \vee [c.1 \wedge c.0 \wedge out.1] \\
Out.0_2 &= [T(i_1.1, \ldots, i_k.1) \wedge c.1] \vee [T_D(i_1.0, \ldots, i_k.0) \wedge c.1 \wedge out.0] \vee [c.1 \wedge c.0 \wedge out.0] \\
Out.c_2 &= \mathbf{0} \\
\\
Out.1_1 &= c.0 \vee [T_D(i_1.0, \ldots, i_k.0) \wedge c.1 \wedge out.1] \vee [c.1 \wedge c.0 \wedge out.1] \\
Out.0_1 &= [T(i_1.1, \ldots, i_k.1) \wedge c.1] \vee [T_D(i_1.0, \ldots, i_k.0) \wedge c.1 \wedge out.0] \vee [c.1 \wedge c.0 \wedge out.0]
\end{aligned}$$

giving final results

$$Out.1 = c.0 \vee [T_D(i_1.0, \ldots, i_k.0) \wedge c.1 \wedge out.1] \vee [c.1 \wedge c.0 \wedge out.1]$$

$$Out.0 = [T(i_1.1, \ldots, i_k.1) \wedge c.1] \vee [T_D(i_1.0, \ldots, i_k.0) \wedge c.1 \wedge out.0] \vee [c.1 \wedge c.0 \wedge out.0]$$

Once again the stored charge on the internal nodes of N will have no effect on the steady state response of `out`, and hence the formulas for these node variables can be eliminated.

These formulas appear more complex than the previous ones, but in fact only require seven binary operations beyond the number required to represent the formulas for T and T_D. Hence, for the series-parallel case, the analyzer has a performance with α slightly more than 2. To better understand these formulas, consider the effect of a sequence in which the clock `c` is first set to 0 and then to 1. The first setting would give a steady state response $Out.1 = \mathbf{1}$ and $Out.0 = \mathbf{0}$. Letting these be the values of $out.1$ and $out.0$, respectively, the second setting would give a steady state response with $Out.1 = T_D(i_1.0, \ldots, i_k.0)$ and $Out.0 = T(i_1.1, \ldots, i_k.1)$. Hence, a domino gate has the same functionality as a static gate,

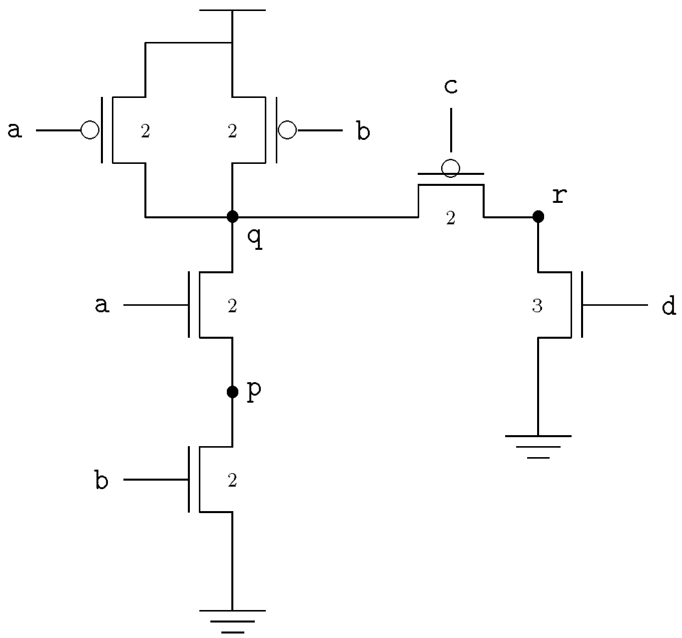

Figure 3: **Example CMOS Circuit.** Transistors are labeled by strength. All storage nodes have size 1. Although rather contrived, this circuit demonstrates a variety of interesting switch-level effects.

even when some inputs equal X. The same result occurs when simulating the gate in ternary mode[31], with `c` set first to 0, then to X, and finally to 1. This indicates that the gate is not sensitive to the rise time of the clock.

In summary, the analyzer performs very well for most logic gates, deriving formulas that correspond directly to the series-parallel structure of the pulldown and pullup networks. Furthermore, even in subnetworks containing logic gates connected to more complex circuitry through pass transistors, the gate functions will be extracted efficiently. This performance should not seem extraordinary, given that logic gates form a particularly simple class of MOS circuits. However, this performance far exceeds the exponential complexity achieved by other general analysis methods. Having the general analysis algorithm obtain efficient results for straightforward cases eliminates the need to devise specialized code to handle these cases.

6.2 Circuit with Stored Charge

Figure 3 shows a small, but relatively complex circuit in terms of its set of possible behaviors. This example demonstrates such properties as complementary logic, bidirectional pass transistors, stored charge, and ratio effects. It is not intended to demonstrate good circuit design practice. The circuit contains a two input NAND gate with output `q`. The gate output is connected by a p-type pass transistor to a node `r` which also has a "kill" transistor to ground. The kill transistor has strength 3, indicating that it can override any

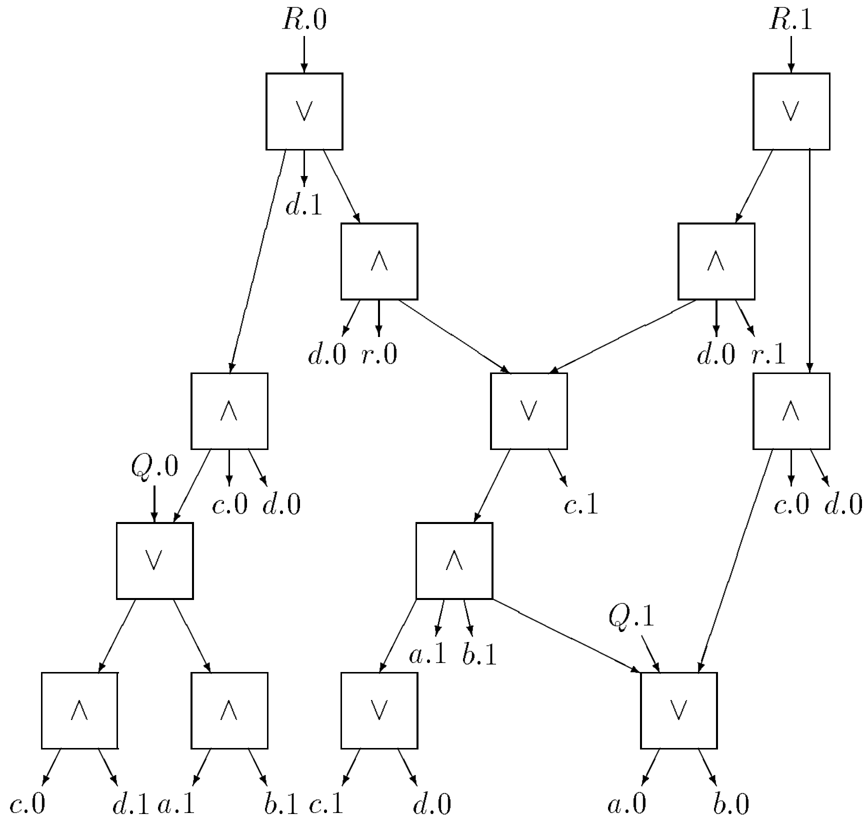

Figure 4: **DAG Representation of Example Circuit**. The leaves denote variables describing the initial state of the circuit, while the vertices denote Boolean operations. The pointers labeled $Q.1$, $Q.0$, $R.1$, and $R.0$ denote formulas for the new states of nodes `q` and `r`.

value transmitted to `r` through the pass transistor. Furthermore, if both the kill and pass transistors are turned on when the gate is driving toward 1, a fight will develop at `q` giving X as its steady state response.

The steps in the analysis are too complex to show in detail. Executing them and simplifying the formulas by hand yields the following results for node `q`:

$$\begin{aligned} Q.1 &= a.0 \lor b.0 \\ Q.0 &= a.1\,b.1 \lor c.0\,d.1. \end{aligned}$$

For clarity, the $\land$ symbols have been removed in the above formulas, and parentheses have been omitted where possible with the convention that AND takes precedence over OR. These formulas are simply those of a NAND gate with an additional term indicating the conditions under which the kill and pass transistor can drive `q` toward 0.

For `r` the analyzer yields the formulas:

$$\begin{aligned} R.1 &= d.0\left[c.1\,r.1 \lor (a.0 \lor b.0)c.0 \lor (a.0 \lor b.0)\,a.1\,b.1\,r.1\right] \\ R.0 &= d.1 \lor d.0\left[c.1\,r.0 \lor a.1\,b.1\,c.0 \lor (a.0 \lor b.0)\,a.1\,b.1\,r.0\right] \end{aligned}$$

These formulas appear quite complex, but each term can be recognized as describing a specific contribution to the steady state response. Part of this complexity owes to the fact that they describe the circuit behavior with some inputs equal to X as well as 0 and 1. From these formulas one can determine the conditions leading to the three possible steady state responses on `r`. The steady state will equal 0 for the following combinations of input and state variables, where a dash indicates that the corresponding variable can equal 0, 1, or X:

a	b	c	d	r
–	–	–	1	–
–	–	1	–	0
1	1	0	–	–
1	1	–	–	0

Similarly, it will equal 1 for the following combinations of input and state variables:

a	b	c	d	r
–	–	1	0	1
0	–	0	0	–
–	0	0	0	–
0	–	–	0	1
–	0	–	0	1

All other cases yield X.

Finally, the formulas for `p` are unimportant. Being an interconnect node within a logic gate, this node is not essential.

Figure 4 shows the DAG representation of the formulas for the steady state response on nodes `q` and `r`. This DAG has 13 nodes, representing 18 binary operations, giving an α of 3. Thus, even a more complex structure in terms of its switch-level behavior has a reasonably concise Boolean description. The size seems especially reasonable considering that the formulas describe the circuit outputs for all 729 ternary combinations of the 4 input and 2 state variables, not just the 64 Boolean combinations.

7 Extensions

Some applications of the switch-level model require features beyond the basic behavioral representation developed so far. Many of these extensions can be provided in straightforward way by modifying the Boolean equations for the steady state response. The ease with which these extensions are incorporated further demonstrates the strength of the mathematical framework.

7.1 Fault Modeling

Switch-level fault simulators such as FMOSSIM [3, 4] have proved very successful at realistically modeling a wide range of faults in MOS circuits. These programs can represent the effects of faults such as nodes forced to ground or supply, as well as transistors stuck open or closed without changing the basic logic model. Normally, the analyzer produces formulas that cannot model these fault effects. Simply injecting faults into the formulas would create faults for which there are no counterparts in the switch-level network as well as overlook faults that could exist in the network. However, a modified analysis can preserve the fault behavior of the circuit. This modified analysis requires only a small number of additional algebraic operations (4 per transistor and 2 per node), although the resulting formulas cannot be simplified as well.

The analyzer injects faults by introducing additional Boolean variables, where a variable is set to 1 when the fault is present and to 0 otherwise. This "fault variable" approach can describe fault effects by Boolean operations and hence apply the solution and manipulation techniques already developed.

For each node `n`, fault variable $n.@$ indicates whether the node acts as an input or a storage node. As an input node, it is stuck at either 0, 1, or X depending on the values of $n.1$ and $n.0$. The analyzer incorporates this variable into the analysis by redefining the terms representing signals of input strength:

$$
\begin{aligned}
N.1_w &= n.@ \wedge n.1, \\
N.0_w &= n.@ \wedge n.0, \\
N.c_w &= \neg n.@.
\end{aligned}
$$

The remainder of the analysis proceeds as before.

For each transistor t, variables $t.@0$ and $t.@1$ indicate the conditions when the transistor is stuck-open (nonconducting) or stuck-closed, respectively. Of course, a stuck-closed fault on a d-type transistor has no effect and hence can be omitted. When a transistor is stuck-closed, it is modeled at its nominal strength, although this can easily be generalized to different strengths. The analyzer incorporates these variables into the analysis by simply modifying the definitions for *indefinite*(t) and *potential*(t) as follows:

type	*indefinite*(t)	*potential*(t)
n-type	$(n.0 \wedge \neg t.@1) \vee t.@0$	$(n.1 \vee t.@1) \wedge \neg t.@0$
p-type	$(n.1 \wedge \neg t.@1) \vee t.@0$	$(n.0 \vee t.@1) \wedge \neg t.@0$
d-type	$t.@0$	$\neg t.@0$

All other steps of the analysis remain unchanged.

The analyzer can model other classes of faults, such as a bridges and breaks in the wires, by introducing additional fault variables. However, its efficiency degrades as the number of fault effects grows large, especially those effects that force a merging of subnetworks.

7.2 Restoring Logic

Conventionally, switch-level simulators ignore voltage degradations through pass transistors caused by threshold effects. This can cause a significant class of design errors to remain undetected. Many CMOS circuits, for example, are designed with the intention that only signals equal to either the supply or ground voltage act as valid logic values. A more conservative model would enforce this rule by yielding X when a 1 passes through an n-type or a 0 passes through a p-type transistor. In cases such as a transmission gate where each signal also passes through a complementary transistor, a 1 or 0 should result. The simulator MOSSIM II [34] optionally enforces such a rule. Yoeli and Brzozowski have proposed a similar rule in their switch-level model [35].

We can incorporate this convention into the switch-level model by modifying the definition of a definite path and consequently the way the analyzer computes $N.c_s$. That is, a path p with $Root(p) = \mathtt{n}$ is definite if either $n = 1$ (respectively, 0) and no transistor in $Trans(p)$ has state X or is of n-type (resp., p-type). In other words, only fully restored signals transmitted through fully conducting transistors can block weaker paths. Note that we need not be concerned about definite paths originating at a node with state X, because if such a path exists, and there is no definite stronger path, then the steady state response will equal X anyhow.

The computation of $N.c_s$ must compute the effects by sources of 1 and 0 separately and combine the two results:

$$N.c_s(n) \quad = \quad N.c1_s \wedge N.c0_s,$$

where formula $N.c1_s$ (respectively, $N.c0_s$) describes the absence of a definite path to node $\mathtt{n}$ originating at a node with state 1 (resp., 0) and having strength greater than or equal to s. the analyzer computes these two values by solving dual systems $[Indef1_s, initc1_s]^D$ and $[Indef0_s, initc0_s]^D$. To formulate these dual systems, define $indefinite1(t)$ and $indefinite0(t)$ for transistor t as follows:

type	$indefinite1(t)$	$indefinite0(t)$
n-type	$\mathbf{1}$	$n.0$
p-type	$n.1$	$\mathbf{1}$
d-type	$\mathbf{0}$	$\mathbf{0}$

That is, $indefinite1(t)$ (respectively, $indefinite0(t)$) describes the cases where the transistor cannot be part of a definite path originating at a node with state 1 (respectively, 0), where **1** indicates "always", and **0** indicates "never".

Edge labeling $Indef1_s$ is defined as $Indef1_w(\mathtt{m},\mathtt{n}) = \mathbf{1}$ and for $w > s > 1$ as

$$Indef1_s(\mathtt{m},\mathtt{n}) \quad = \quad Indef1_{s+1}(\mathtt{m},\mathtt{n}) \wedge \bigwedge_{t \in \mathcal{T}_s(\mathtt{m},\mathtt{n})} indefinite1(t).$$

Vertex labeling $initc1_s$ is defined for storage node $\mathtt{n}$ and strength $w > s > 1$ as

$$initc1_s(\mathtt{n}) \quad = \quad \begin{cases} N.c1_{s+1} \wedge \bigwedge_{\mathtt{m} \in \mathcal{N}_w} \bigwedge_{t \in \mathcal{T}_s(\mathtt{m},\mathtt{n})} [indefinite1(t) \vee m.0] & s > k \\ n.0 & \mathtt{n} \in \mathcal{N}_s \\ N.c1_{s+1} & \text{else} \end{cases}$$

with the convention that $N.c1_w = \mathbf{1}$. Observe how this formula uses variables $n.0$ and $m.0$ to place restrictions on the root node state. If $n.0 = 1$, then node $\mathtt{n}$ cannot be the root of a definite path p with state 1, and similarly for $m.0$.

Similarly, edge labeling $\mathit{Indef0}_s$ is defined as $\mathit{Indef0}_w(\mathtt{m},\mathtt{n}) = \mathbf{1}$, and for $w > s > 1$ as

$$\mathit{Indef0}_s(\mathtt{m},\mathtt{n}) \quad = \quad \mathit{Indef0}_{s+1}(\mathtt{m},\mathtt{n}) \wedge \bigwedge_{t \in \mathcal{T}_s(\mathtt{m},\mathtt{n})} \mathit{indefinite0}(t).$$

Vertex labeling $\mathit{initc0}_s$ is defined as

$$\mathit{initc0}_s(\mathtt{n}) \quad = \quad \begin{cases} N.c0_{s+1} \wedge \bigwedge_{\mathtt{m} \in \mathcal{N}_w} \bigwedge_{t \in \mathcal{T}_s(\mathtt{m},\mathtt{n})} [\mathit{indefinite0}(t) \vee m.1] & s > k \\ n.1 & \mathtt{n} \in \mathcal{N}_s \\ N.c0_{s+1} & \text{else} \end{cases}$$

with the convention that $N.c0_w = \mathbf{1}$.

This extension adds an extra system of equations at each level and hence results in somewhat larger formulas. This seems a reasonable price to pay for detecting an additional class of circuit design errors.

As an example, applying this modified analysis to the circuit of Figure 3 has no effect on node `q`, but for node `r` yields the formulas:

$$\begin{aligned} R.1 &= d.0 \left[c.1\, r.1 \;\vee\; (a.0 \vee b.0) c.0 \;\vee\; a.1\, b.1\, r.1 \right] \\ R.0 &= d.1 \;\vee\; d.0 \left[c.1\, r.0 \;\vee\; a.1\, b.1\, c.0 \;\vee\; a.1\, b.1\, r.0 \right] \end{aligned}$$

This reduces the cases for which `r` has steady state response 0 to the following:

a	b	c	d	r
–	–	–	1	–
–	–	1	–	0
1	1	–	–	0

That is, `r` will not be pulled to 0 when $a = b = 1$ and $c = 0$, unless $r = 0$ or $d = 1$.

7.3 Charge Decay

During normal operation, most switch-level simulators assume that a node retains its stored charge indefinitely. In actual circuits leakage currents cause stored charge to eventually decay to some indeterminate value. A simulator that does not model this decay will fail to detect cases in which proper behavior depends on stored charge being maintained beyond some reasonable time limit.

The simulator MOSSIM II has an optional mode in which charge is retained only for a number of clock cycles specified by the user. Any storage node that remains unrefreshed for this many cycles is set to X. Such an event does not in itself indicate a circuit design error. However, any further operations that depend on this node state will yield more X values, and hence invalid uses of stored charge can be detected. MOSSIM II implements this feature by tagging every node with the most recent refresh time, measured in clock cycles. Due to subtleties caused by both charge sharing and by transistors in the X state, it must use a

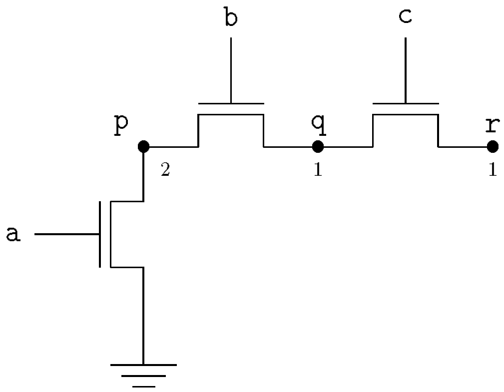

Figure 5: **Charge Sharing Circuit Example.** Nodes `q` and `r` can obtain new logic values by sharing charge with `p`.

rather complex algorithm to compute the effective refresh time of a node as it updates the state.

The capability provided by MOSSIM II cannot be described efficiently in terms of Boolean operations, because every node state must specify both a logic value and an integer refresh time. By adopting a somewhat less precise timing scheme, however, a Boolean representation becomes more practical. In this scheme, the period over which the circuit operates is divided into a series of "epochs". An epoch will typically have duration equal to half the maximum number of clock cycles for which stored charge may be assumed valid. A flag is maintained for each storage node indicating whether the node is "fresh" or "stale", i.e., whether or not it has been refreshed during the current epoch. This flag is updated every time the node state is recomputed. At the end of each epoch the states of all stale nodes are set to X. At the same time, all nodes not currently connected to input nodes are marked as stale for the start of the next epoch. With this scheme, the exact charge retention time can range between just over one epoch to just under two, depending on the alignment between the refresh time and epoch boundaries. This degree of accuracy suffices for most applications, because most designers set conservative limits on charge retention time.

Developing a precise definition of the conditions under which a node is refreshed involves several subtleties. Clearly, a node is refreshed whenever it is connected to an input node by a set of transistors in the 1 state. Consider, however, the circuit shown in Figure 5 in which several storage nodes may share charge. In particular, node `p` has greater size than either `q` or `r`, and hence the state of `p` can override those of `q` and `r`. In some circuit designs, such as where `p` is high capacitance bus, the transistors may be operated in such a way that no conducting path ever forms between an input node and either `q` or `r`. Such cases can be handled with a convention that whenever nodes `q` or `r` share charge with `p`, they will be marked as fresh if `p` is fresh and as stale otherwise. As a further subtlety, when transistors in the X state are present, a node may or may not be refreshed depending whether or not this transistor is actually conducting. Such cases can be handled with a convention that a node should be marked as stale if its state may depend on that of some stale node for some combination of conducting and nonconducting transistors. To summarize these conventions precisely, a node should be marked as stale whenever it is the destination of an unblocked path originating at a stale node.

To express these conditions symbolically, the analyzer introduces a variable $n.*$ for every storage node `n`, with value 1 indicating "stale" and 0 indicating "fresh". The analyzer then generates a formula $N.*$ for each node specifying when it should be marked as stale as a function of the state and refresh variables associated with the nodes. The analyzer generates these formulas in a manner similar to that used to generate the formulas $N.1$ and $N.0$. Starting at strength $s = k$ (the maximum storage node size), and working downward to 1, it solves the system $[Conduct_s, init*_s]$ to generate formulas $N.*_s$ for each node `n`. The desired formula $N.*$ equals $N.*_1$. The edge labeling $Conduct_s$ has already been defined as Equation 10. The vertex labeling $init*_s$ indicates the conditions under which each node may be the source of stale charge:

$$init*_s(\mathtt{n}) \quad = \quad \begin{cases} N.*_{s+1} \;\vee\; [N.c_{s+1} \wedge n.*] & \mathtt{n} \in \mathcal{N}_s \\ N.*_{s+1} & \text{else,} \end{cases}$$

with the convention that $N.*_{k+1} = \mathbf{0}$.

The simulator utilizes these formulas as follows. At the end of an epoch, the simulator makes two passes over the nodes. The first pass sets the state of any node `n` for which $n.* = 1$ to X and also sets $n.*$ to 1. The second pass evaluates the formula $N.*$ to determine the new value of $n.*$. This second pass marks as fresh only nodes to which all unblocked paths originate at input nodes. As the simulator proceeds, every time it computes a new state for node `n`, it computes a new value of $n.*$ by evaluating the formula $N.*$.

As an example, applying this analysis to the circuit of Figure 5 yields the following formulas:

$$\begin{aligned} P.* &= a.0\, p.* \\ Q.* &= a.0\, b.1\, p.* \;\vee\; b.0\, q.* \;\vee\; b.0\, c.1\, r.* \\ R.* &= a.0\, b.1\, c.1\, p.* \;\vee\; b.0\, c.1\, q.* \;\vee\; b.0\, r.* \;\vee\; c.0\, r.*. \end{aligned}$$

Observe that `p` can be marked as stale only if it is already stale and $a \neq 1$. On the other hand, `q` is marked as stale if it shares charge with a stale value on either `p` or `r`, or if it is isolated and already stale. Similar results hold for `r`.

8 Conclusion

Transforming a switch-level network into an explicit functional representation has proved a challenging task. Previous attempts yielded results that were too inefficient or too inaccurate for practical use. The solution presented here relies on three major ideas. First, systems of Boolean equations can describe switch-level networks. Second, Gaussian elimination can take advantage of the sparse structure of these systems and generally give solutions of linear complexity. Finally, the DAG representation of a set of formulas can exploit the sharing of common subexpressions to give a very compact result.

The analyzer has the potential to improve the efficiency of programs for a variety of MOS circuit analysis tasks. It can incorporate a number of modeling extensions by modifying or augmenting the system equations. The advantage of this approach to switch-level modeling will increase as hardware becomes available that achieves high performance through greater degrees of specialization and concurrency.

References

[1] R. E. Bryant, "A Switch-Level Model and Simulator for MOS Digital Systems," *IEEE Trans. on Computers* Vol. C-33, No. 2 (February, 1984) pp. 160–177.

[2] C. J. Terman, *Simulation Tools for Digital LSI Design*, PhD Thesis, MIT Dept. Elec. Eng. and Comp. Sci. (October, 1983).

[3] M. D. Schuster, and R. E. Bryant, "Concurrent Fault Simulation of MOS Digital Circuits", *Advanced Research in VLSI*, P. Penfield, Jr., *ed.*, MIT (1984), pp. 160–177.

[4] R. E. Bryant, and M. D. Schuster, "Performance Evaluation of FMOSSIM, a Concurrent Switch-Level Fault Simulator", *22nd Design Automation Conf.*, ACM (1985), pp. 715–719.

[5] H. H. Chen, R. G. Mathews, and J. A. Newkirk, "An Algorithm to Generate Tests for MOS Circuits at the Switch-Level", *International Test Conf.*, IEEE (1985).

[6] M. K. Reddy, S. M. Reddy, and P. Agrawal, "Transistor Level Test Generation for MOS Circuits", *22nd Design Automation Conf.*, ACM (1985) pp. 825–828.

[7] R. E. Bryant, "Symbolic Verification of MOS Circuits," *1985 Chapel Hill Conf. on VLSI,* H. Fuchs, *ed.* Computer Science Press (1985), pp. 419–438.

[8] D. S. Reeves, and M. J. Irwin, "Functional Verification of Digital MOS Circuits", *IEEE International Conf. on Computer-Aided Design*, (1986), pp. 496–499.

[9] E. Ulrich, and T. Baker, "The Concurrent Simulation of Nearly Identical Digital Networks", *IEEE Computer* (April, 1974), pp. 39–44.

[10] M. M. Denneau, "The Yorktown Simulation Engine", *19th Design Automation Conf.*, ACM (1982), pp. 55–59.

[11] *Daisy Megalogician Product Description*, Daisy Systems, 1984.

[12] *ZyCad LE-001 and LE-002 Product Description*, ZyCad Corp., 1982.

[13] W. J. Dally and R. E. Bryant, "A Hardware Architecture for Switch-Level Simulation", *IEEE Trans. on Computer-Aided Design of Integrated Circuits*, Vol. CAD-4, No. 3 (July, 1985), pp. 239–249.

[14] E. H. Frank, "Switch-Level Simulation of VLSI Using a Special-Purpose, Data-Driven Computer", *22nd Design Automation Conf.*, ACM (1985) pp. 735–738.

[15] G. Pfister, private communication, 1980.

[16] Z. Barzilai, *et al*, "Simulating Pass Transistor Circuits Using Logic Simulation Machines", *19th Design Automation Conf.*, ACM (1983), pp. 157–163.

[17] J. Hayes, "A Unified Switching Theory with Applications to VLSI Design", *Proc. IEEE*, Vol. 70, No. 10 (October, 1982), pp. 1140–1151.

[18] Z. Barzilai, *et al*, "SLS—a Fast Switch Level Simulator for Verification and Fault Coverage Analysis", *23rd Design Automation Conf.*, ACM (1986), pp. 164–170.

[19] E. Cerny, and J. Gecsei, "Simulation of MOS Circuits by Decision Diagrams", *IEEE Trans. on Computer-Aided Design of Integrated Circuits*, Vol. CAD-4, No. 4 (October, 1985), pp. 685–693.

[20] G. Ditlow, W. Donath, and A. Ruehli, "Logic Equations for MOSFET Circuits", *International Symposium on Circuits and Systems*, IEEE (1983), pp. 752–755.

[21] I. N. Hajj, and D. Saab, "Symbolic Logic Simulation of MOS Circuits", *International Symposium on Circuits and Systems*, IEEE (1983).

[22] C. A. Mead, and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley, 1980.

[23] M. R. Garey, and D. S. Johnson, *Computers and Intractability*, Freeman, 1979.

[24] L. W. Nagel, *SPICE2: A Computer Program to Simulate Semiconductor Circuits*, PhD Thesis, Univ. of California, Berkeley, Dept. of Elec. Eng., 1975.

[25] R. E. Bryant, *Algorithmic Aspects of Symbolic Switch Network Analysis*, companion paper, 1987.

[26] C. Lutz, S. Rabin, C. Seitz, and D. Speck, "Design of the MOSAIC Element," *Advanced Research in VLSI*, P. Penfield, Jr., *ed.*, MIT (1984), pp. 1–10.

[27] R. Byrd, G. D. Hachtel, and M. R. Lightner, *Switch Level Simulation: Part I—Theory and Algorithmic Frame*, unpublished, 1985.

[28] M. Yoeli, and S. Rinon, "Application of Ternary Algebra to the Study of Static Hazards," *J.ACM*, Vol. 11, No. 1 (January, 1964), pp. 84–97.

[29] L. A. Glasser, and D. W. Dobberpuhl, *The Design and Analysis of VLSI Circuits*, Addison-Wesley, 1985.

[30] N. H. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley, 1985.

[31] R. E. Bryant, "Race Detection in MOS Circuits by Ternary Simulation", *VLSI83*, F. Anceau and E. J. Aas, *ed.* North-Holland (1983), pp. 85-95.

[32] M. Takashima, *et al*, "Programs for Verifying Circuit Connectivity of MOS/LSI Mask Artwork", *19nd Design Automation Conf.*, ACM (1982), pp. 544–550.

[33] C. Ebeling, and O. Zajicek, "Validating VLSI Circuit Layout by Wirelist Comparison", *IEEE International Conf. on Computer-Aided Design*, (1982), pp. 172–173.

[34] R. E. Bryant, M. D. Schuster, and D. Whiting, *MOSSIM II: A Switch-Level Simulator for MOS LSI, User's Manual*, Technical Report 5033, Dept. of Comp. Sci., Caltech (1982).

[35] M. Yoeli, and J. A. Brzozowski, "A Mathematical Model of Digital CMOS Networks", *Canadian Conf. on VLSI* (1985).