A Switch-Level Simulation Model
for
Integrated Logic Circuits

by

Randal Everitt Bryant

© Massachusetts Institute of Technology, 1981

March, 1981

This research was supported in part by the Department of Energy under contract number DE-AC02-79ER10473, and in part by the United States Air Force under contract number AFOSR F49620-80-C-0073.

Massachusetts Institute of Technology
Laboratory for Computer Science

Cambridge **Massachusetts 02139**

This blank page was inserted to preserve pagination.

A Switch-Level Simulation Model for Integrated Logic Circuits

by

Randal Everitt Bryant

© Massachusetts Institute of Technology, 1981

March, 1981

This research was supported in part by the Department of Energy under contract number DE-AC02-79ER10473, and in part by the United States Air Force under contract number AFOSR F49620-80-C-0073.

Massachusetts Institute of Technology
Laboratory for Computer Science
Cambridge **Massachusetts 02139**

A Switch-Level Simulation Model for Integrated Logic Circuits

by

Randal Everitt Bryant

Submitted to the Department of Electrical Engineering and Computer Science on March 31, 1981
in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

Abstract

Switch-level simulators model a metal oxide semiconductor (MOS) large scale integrated (LSI) circuit as a network of transistor "switches". They can simulate many aspects of MOS circuits which cannot be expressed in the Boolean logic gate model, such as bidirectional pass transistors, dynamic storage, and charge sharing. Furthermore, the logic network can be extracted directly from the mask specification by a relatively straightforward computer program. Unlike analog circuit simulators, however, the nodes are assigned discrete states 0, 1, and X (for unknown), and the transistors are assigned discrete states "open", "closed", and "unknown". As a consequence, switch-level simulators operate at speeds comparable to logic gate simulators.

In this thesis, a formal model of switch-level networks is developed. The networks in this model may contain transistors of different strengths and types, as well as nodes of different sizes and types, and hence the logical behavior of a wide variety of ratioed, complementary, and ratioless designs can be expressed. In keeping with the concept of a *logic* model, however, both the transistor strengths and the node sizes may take on only discrete values, and electrical behavior is modeled in a highly idealized way. The operation of a network is characterized by its *target state* function, which for a particular state of the network yields the logic states which the nodes would eventually reach if all transistors were held fixed in their initial states. This characterization abstracts away the rate at which nodes approach their target states and the voltages through which they pass but provides adequate detail for many simulation and analysis techniques. The target state function can be defined in terms of an abstraction called *logic signals*, where a logic signal gives a composite description of the network at some node much as a Thevenin equivalent network gives a composite description of a linear network at some port.

Logic signals can be formalized into a simple, discrete algebra with operations describing the effects of performing some elementary network transformations. A technique for finding the target state of an arbitrary switch-level network can be derived by utilizing concepts from abstract algebra and lattice theory. This technique leads to an algorithm for a switch-level simulator which improves on previous algorithms in its generality, speed, and simplicity. Furthermore, the mathematical formulation provides a means for proving useful properties about the simulation method and opens up further areas of application for the switch-level model.

Thesis Supervisor: Jack B. Dennis

Title: Professor of Electrical Engineering and Computer Science

Keywords: Switch-level simulation, logic simulation, computer-aided design, large scale integration.

Acknowledgments

I would like to thank Professor Jack B. Dennis for his continued intellectual and financial support during my graduate studies. He has taught me the value of defining a problem in terms of a set of simple, but often untraditional concepts upon which the solution can be based. Other present and past members of the Computation Structures Group have greatly contributed to this environment including Professor Arvind, Clement Leung, Dean Brock, Bill Ackerman, Andy Boughton, Narinder Singh, Sheldon Borkin, and Ken Weng.

My introduction to integrated circuit design was provided by Ms. Lynn Conway of Xerox PARC, and she has provided valuable encouragement of my research efforts. As a Teaching Assistant for Professor Jonathon Allen, I began serious work in this area, and his course provided a test facility for my simulator. The students in this course showed remarkable patience and enthusiasm. Both Professor Allen and Professor Paul Penfield have provided continued interest and encouragement in my work and served as readers for this thesis.

Much of the development of switch-level simulation has occurred through the efforts of Chris Terman and Clark Baker. My work has benefitted greatly from their ideas and experiences. The VLSI design communities at MIT and at other institutions have also provided a strong, motivating force for these developments.

The following people also assisted me by reading earlier drafts of this thesis: Dean Brock, Chris Terman, Clark Baker, and Wayne Gramlich.

This research was supported in part by the Department of Energy under contract number DE-AC02-79ER10473, and in part by United States Air Force Contract AFOSR F49620-80-C-0073.

Contents

1. Introduction 7

1.1 The Boolean Gate Model 10
1.2 Analog and Hybrid Simulators 14
1.3 Switch-Level Simulators 15
1.4 An Abstract Model for MOS LSI 17
1.5 Relation to Relay Networks 20
1.6 Outline of Thesis 21
1.7 Notation 23

2. The Switch-Level Network Model 24

2.1 Introduction 24
2.2 Network Structure 24
2.3 Network Representation 29
2.4 Logic States 29
2.5 Network State 30
2.6 The Target State Function 31
2.7 Specification of the Target State 33
2.8 Relation to Actual Circuits 39
2.9 Comparison to Other Switch-Level Models 40
2.10 Derivation of the Switch-Level Network 43
2.11 Summary 45

3. Logical Conductance Networks 46

3.1 Introduction 46
3.2 Properties of the Electrical Model 46
3.3 Simplification of the Target State Definition 49
3.4 Rational Functions 55
3.5 Equivalent Networks 58
3.6 Logical Conductance 63
3.7 Properties of Logical Conductance Networks 66
3.8 Summary 70

4. Logic Signals 71

4.1 Introduction 71
4.2 Definition 71
4.3 Rules for Logic Signals 74
4.4 The Steady State Signals 83
4.5 Constraints on the Steady Steady State Signals 85
4.6 Specification of the Steady State Signals 88
4.7 Signal Blocking 93
4.8 Conclusion 94

5. An Algebra of Logic Signals 96

5.1 Introduction 96
5.2 General Definitions 96
5.3 The Algebra of Signal Strengths 97
5.4 The Algebra of Signal States 101
5.5 The Algebra of Signals 103
5.6 Summary 112

6. Computation of the Target State 116

6.1 Introduction 116
6.2 The Target State Equation 116
6.3 The Steady State Signal Equation 119
6.4 Solution for Restricted Logical Conductance Networks 121
6.5 Solution for General Logical Conductance Networks 125
6.6 The Target State 133
6.8 Properties of the Target State 138
6.9 Summary 141

7. Simulation Algorithms 142

7.1 Introduction 142
7.2 Complexity Model 143
7.3 Sparse and Incremental Equations 145
7.4 Unit Delay Simulation Algorithm 153
7.5 Pseudo Unit Delay Simulation Algorithm 157
7.6 Comparison to Other Switch-Level Simulators 158
7.7 Mixed-Level Simulation 164
7.8 Performance of MOSSIM 166
7.9 Summary 169

8. Timing Models 170

8.1 Introduction 170
8.2 Simulation Timing Models 173
8.3 Analysis of Timing by Ternary Simulation 178
8.4 Toward a Simulator for Self-Timed Systems 183
8.5 Summary 185

9. Conclusions 186

9.1 Final Thoughts 186
9.2 Suggestions for Further Research 187

Appendix I. Multi-Port Networks 194

I.1 Introduction 194
I.2 Port Parameters 195
I.3 The Effect of a Variable Resistor 197
I.4 The Effect of Connecting Two Ports 200

Appendix II. Proofs of Results in Chapter 6 202

II.1 Passive Networks 202
II.2 Theorem 6.4 206
II.3 Corollary 6.4.1 208
II.4 Theorem 6.5 210
II.5 Theorem 6.7. 214

References 216

Biographical Note 219

1. Introduction

Recently, a new class of logic simulator has emerged specifically for simulating metal oxide semiconductor (MOS) large scale integrated (LSI) circuits. These *switch-level* simulators model an MOS design as a set of nodes connected by transistor "switches" with each node assuming a state 0, 1, or X (unknown) and each switch a state "open", "closed", or "unknown". Programs such as the author's MOSSIM [8, 9] and others [5] show remarkable accuracy and versatility in simulating such logic elements as logic gates, pass transistor logic, busses, and both static and dynamic memory. The accuracy results because the logic network closely matches the actual circuit, while the versatility results because transistors form a common denominator for all LSI design techniques. Furthermore these simulators operate at sufficient speeds to test entire LSI systems, because behavior is modeled at a logical rather than a detailed electrical level. Unlike previous attempts at developing MOS logic simulators by adding *ad hoc* extensions to gate-level simulators, switch-level simulators are based on a uniform and consistent model which provides a powerful level of abstraction for viewing MOS designs. In this thesis the concept of switch-level simulation is developed into a mathematical model of MOS logic networks from which simulation algorithms and other analytic tools can be derived.

The ability to implement digital logic has progressed greatly in the past decade with circuits of increasing size and complexity being fabricated at decreasing cost. Metal oxide semiconductor (MOS) technology has played a major role in this "integrated circuit revolution" due to its relative simplicity in both design and fabrication. In more recent years MOS design has become part of the university curriculum in both electrical engineering and computer science. By using simplified design rules and conservative clocking schemes, and by following systematic methodologies such as those presented by Mead and Conway [28], the basics of MOS design can be learned in one semester. As this training becomes widespread, we will see a new form of integrated circuit revolution in which nonspecialists design their own custom integrated circuits rather than relying on the limited variety of commercially

available products.

With the increasing number of custom-designed integrated circuits, and with the growing size and complexity of commercial LSI products, the inadequacy of current LSI design techniques has become apparent. The semiconductor industry has traditionally relied on humans to design, lay out, and verify LSI systems. Typically many man hours are spent, and several prototype chip designs are fabricated in developing a single IC design. Industry analysts have extrapolated the current design techniques and estimated that a 100,000 device microprocessor (the expected state of the art in 1982) would take 60 man-years to lay out and another 60 to debug [41]. Rather than accepting such predictions as inevitable, a change in design techniques is called for.

Computerized tools have been applied to commercial LSI design, but most of these can be viewed as extensions of manual techniques (such as graphical layout systems), or as experts in a specialized domain (such as circuit simulators.) Both kinds require close cooperation with a human who understands the exact capabilities and limitations of the program. In addition, humans are required to bridge the gaps between programs with expertise in different domains. For example, before a design can be tested by a logic simulator, the actual design typically must be translated by hand into a description which can be understood by the simulator. This translation process wastes manpower in performing a rather tedious task and also decreases the level of confidence provided by the simulation.

Logic simulators form an important class of computerized tools for LSI design. Their utility has long been recognized for analyzing designs which by their size and complexity exceed the capability of humans to fully understand. The usefulness of a logic simulator depends greatly on the consistency and accuracy with which it can model the full range of design techniques available to the designer. Of course no logic simulator can model all designs with complete accuracy, because it does not simulate the detailed analog behavior. Nonetheless, it should provide as close a model as possible within a set of well-defined limitations. As a further requirement, a logic simulator for LSI must be efficient enough to simulate entire systems with reasonable speed. The size of single-chip, very large scale integrated (VLSI) systems

will far exceed the small scale integrated (SSI) systems for which conventional logic simulators were designed.

A logic simulator has as its basis an abstract model of how digital systems function. This *logical model* describes both the structure and the behavior of a system in terms of a set of primitive elements, a set of interconnections, and a set of rules for operation. For a simulator to accurately and reliably simulate a system, the logical model must reflect its actual structure and operation.

Unfortunately, the development of logic simulators has not kept pace with LSI technology. This inadequacy stems in part from the lack of formal logic models for describing the behavior of MOS circuits.[1] Instead, systems are designed and simulated using an *ad hoc* combination of Boolean gate models, relay models, and electronic models. Such a representation may be appropriate for human designers, who can combine different modes of operation and resolve the conflicts between these models. Computers, however, lack the intuition required to simultaneously view a system at several different levels. Computerized tools must be based on models which can describe a large class of systems in a more uniform way.

Most logic simulators are based on the Boolean gate model which adequately models systems built from SSI components but fails to support the wide variety of techniques available to the LSI designer, especially for MOS LSI. Many extensions of the Boolean gate model have been attempted with the usual result that only a slightly larger class of designs can be simulated and many sources of unpredictable or inaccurate behavior are introduced. This claim will be supported by briefly surveying the development of logic simulators with respect to their support for MOS LSI design.

---

1. Brzozowski and Yoeli [10] present a logic model in which "T-elements" provide a simplified model of field-effect transistors. This model, however, only expresses the operation of static logic gates. Furthermore, it has not received widespread attention.

1.1 The Boolean Gate Model

The Boolean logic gate model has formed the theoretical basis for logic design ever since the advent of electronic logic. In this model a system consists of a set of logic gates connected by unidirectional, memoryless wires. The logic gates compute Boolean functions of their input signals and transmit these values along the wires to the inputs of other gates. Each gate input has a unique signal source. Information is stored only in the feedback paths of sequential circuits. This model directly implements Boolean algebra [20] and hence has a well-defined specification which can guide the simulator implementation.

The Boolean gate model cannot describe many of the techniques available to the logic designer, especially the MOS LSI designer. MOS pass transistor networks can implement combinational logic in ways which more closely resemble relay contact networks than logic gate networks (see [28] or [16] for several examples.) Dynamic memory can store information without feedback paths by exploiting the capacitances of the wires and the gates of the transistors attached to them. A variety of bus structures can provide multidirectional, multipoint communication. A logic simulator which implements only the Boolean gate model provides limited support to the MOS LSI designer. Most existing logic simulators, however, extend the Boolean gate model in various ways. Hence, any evaluation of logic gate simulators must consider these extensions as well.

Many simulators extend the two-valued logic of Boolean algebra with a third value to represent an unknown or undefined logic level. This "X" level can indicate an uninitialized state variable, a signal held between the two logic thresholds, or a signal in transition between 0 and 1 or between 1 and 0. The X logic level can be handled algebraically by changing the two-valued Boolean algebra to a three-valued DeMorgan's algebra [3, 11, 23, 46].[1] Thus, even with this extension many of the desirable mathematical

---

1. A DeMorgan's Algebra satisfies all postulates of Boolean Algebra except for the Law of Excluded Middle ($A + \neg A = 1$).

properties of the Boolean gate model are preserved. Alternatively, some simulators implement the X level by an enumeration technique in which the simulation is repeated with the nodes at the X level set to all possible combinations of 0's and 1's [6, 45]. Nodes which remain at a unique level for all combinations are set to this level, while all others are set X. Still other simulators [44] use *ad hoc* techniques to implement the X level often resulting in inconsistent or anomalous behavior. The X logic level is useful in simulating all forms of digital logic including MOS.

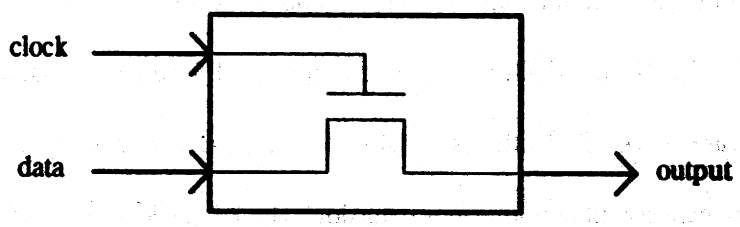
To model the behavior of bus structures, some logic simulators have a fourth or "high-impedance" logic level [12]. This H level corresponds to the third state of tri-state logic. To simulate a bus structure, the outputs of a number of gates are connected to a common node. Typically all but one output will be at the H level, and the level of that output will dominate. Unlike the X level which can be viewed as an extension of Boolean algebra, the H level violates a basic principle of the Boolean model, in that a logic gate input no longer has a unique signal source. Because the simulator is not based on a well-defined mathematical model, it becomes difficult to implement consistently and accurately. The H state may be adequate for simulating SSI designs in which only limited forms of tri-state busses can be implemented. The MOS LSI designer, on the other hand, can select from a wide variety of bus designs, such as pre-charge/discharge and multiple driver designs. The H state only partially captures the behavior of these bus structures. Nonetheless, the H logic level is seen in simulators for both MOS and other forms of logic.

Some simulators allow a special logic gate to represent the MOS pass transistor [44]. This logic gate models a field-effect transistor (FET) as a unidirectional device with two inputs and one output as shown in Figure 1.1. The simulator cannot help in cases in which the bidirectional property of the FET is important, such as in circuits where information may flow in either direction,[1] or where the design has a

---

1. In actual fact, the bidirectional property of the FET is rarely used intentionally. The author has seen only a few designs which have signals propagating in both directions through a pass transistor.

Fig. 1.1. The FET Logic Gate



clock	data	output
0	0	H
0	1	H
1	0	0
1	1	1

malfunction due to a sneak path. More recent gate model simulators have implemented bidirectional transistor models, but these transistors usually entail a much higher computational cost, and hence their use must be minimized. Furthermore, few of these simulators can simulate arbitrary combinational networks of pass transistors, because they cannot model the action of pullup resistors. Thus this extension does not fully capture the behavior of the MOSFET. Like the high-impedance logic level, it also lacks the algebraic properties of the Boolean gate model and hence forces an *ad hoc* implementation.

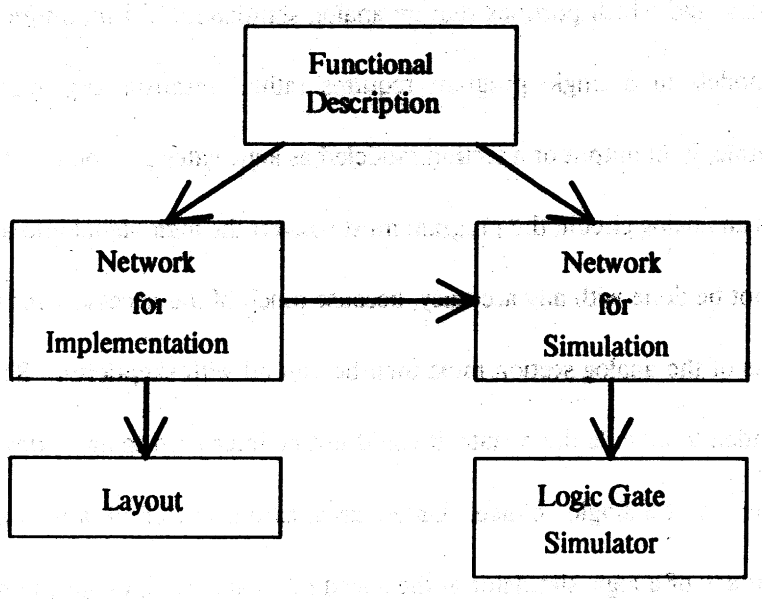
Finally, some simulators model dynamic memory in a limited fashion [44]. A node is allowed to remain at a previous logic level if the outputs of all logic gates connected to the node are at the H level. This extension is very limited in its generality and its accuracy.[1]

As new types of MOS logic circuits are developed, designers add more extensions of the Boolean gate model to their simulators. These extensions are doomed to failure, because they cannot correct the fundamental mismatch between the Boolean gate model and MOS logic circuits. MOS circuits consist of bidirectional switching elements connected by bidirectional wires with memory (considering the capacitive effects of the transistor gates as contributing to a wire's memory.) Instead, the simulators become increasingly cumbersome and unreliable, because they only partially capture the behavior of the logic technology.

1. SIMULOG introduces even greater inaccuracy by failing to differentiate between the undefined (X) level and the high-impedance (H) level.

Using a gate model simulator requires both intuition about how the design is supposed to function and detailed knowledge of the simulator implementation. The user must explicitly identify the logic gates, the signal directions through pass transistors, the locations of busses, the sites of dynamic memory, and sometimes even the feedback paths. Often the actual logic design must be transformed into one compatible with the simulator which may not display the exact same behavior. This transformation process not only decreases the level of confidence provided by the simulation, it virtually eliminates the possibility of automatically generating the simulation network from some specification of the actual design. Unless we restrict our attention to a limited class of designs, a very sophisticated program would be required to analyze the mask patterns for an MOS layout and convert this to a gate-level description, performing the necessary transformations to provide compatibility with the simulator. Without this capability, a logic simulator cannot be used to help verify the correctness of a layout. Gate-level logic simulators fit into the MOS LSI design process as shown in Figure 1.2. They provide mainly a verification of the high level functional description plus limited verification of the actual logic design.

Fig. 1.2. Role of Logic Gate Simulators in LSI Design



1.2 Analog and Hybrid Simulators

LSI designers have recognized the limitations of conventional logic simulators for modeling MOS circuits and have at times resorted to analog or hybrid simulators. Analog simulators treat the entire design as network of analog circuit elements and try to model the detailed waveform at every node over time. Simulators such as SPICE [30] and even those which use faster (and more approximate) numerical techniques such as MOTIS [13] require very large amounts of computation. Some reports claim the amount of computation scales as the square of the network size [2]. Thus they are practical only for small designs or for small sections of larger designs. Analog simulators, however, are based on a uniform and general abstract model and hence have been well received because of their consistency and accuracy. Furthermore, computer programs exist for deriving the simulation network automatically from the layout descriptions [2].

The amount of computation can be reduced significantly by hybrid techniques such as in SPLICE [31] in which some sections of the design are simulated as logic gates and others are simulated as analog circuits. Hybrid simulation works well as long as only small, isolated sections of the design need be simulated as analog circuits. Unfortunately, a human must decide which portions of the network can be modeled as logic gates, and which portions require analog simulation. Furthermore, trying to combine analog and logic models in a single program requires rather unsatisfactory approximations at the interfaces. For example, if an output of a section modeled as logic gates is to be interfaced to an input of a section modeled as an analog circuit, the program must convert the logic signal into a voltage waveform. This, of course, cannot be done with any accuracy, because much of the necessary information is lacking. The resultant outputs of the analog section must then be viewed with skepticism. Similarly, if the logic simulator were extended to include the X state, it could not be interfaced to an analog simulator because this state does not represent a single voltage. Unless great care is exercised, a hybrid simulation could well provide the accuracy of a logic simulator at the speed of an analog simulator, rather than *vice-versa*.

For this reason, a human must monitor the simulation very carefully.

1.3 Switch-Level Simulators

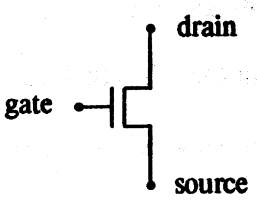
As an alternative to conventional logic simulators, the author has developed the simulator MOSSIM [8, 9] specifically for the logical simulation of MOS LSI. With MOSSIM the Boolean gate concept is discarded altogether and replaced with a logical model which closely matches the structure and behavior of MOS circuits. A logic network consists of a set of nodes connected by a set of FET "switches". MOSSIM uses three logic levels: 0, 1, and X (undefined.) There are three types of nodes:

1. *Input* nodes provide a strong, externally generated signal (*e.g.* power lines, clock drivers, data inputs, etc.)
2. *Pullup* nodes are connected via a pullup resistor to a high voltage. They will generate a 1 signal unless grounded. The output of an nMOS logic gate is an example of a pullup node.
3. *Normal* nodes cannot generate a signal but can store a signal dynamically.

Only two types of network elements are allowed: p-type and n-type field-effect transistors. A transistor is a three node device which acts as a voltage-controlled switch with no assumed direction of signal flow as shown in Figure 1.3. No distinction is made between the labels "source" and "drain". When the gate node of a transistor is in the X state, the switch status is unknown: it may be open, closed, or somewhere between. The user interface of MOSSIM allows the user to describe the network in terms

---

Fig. 1.3. The MOSSIM Transistor Model



p-type		n-type	
gate	effect	gate	effect
0	closed	0	open
1	open	1	closed
X	unknown	X	unknown

of transistors, logic gates, and user-defined macros, but these are all translated into a transistor-level representation for the simulation. C. Terman has developed a switch-level simulator patterned after MOSSIM [5] but differing in several respects, as is discussed at several points in this thesis. Researchers at Caltech [34] also developed a switch-level MOS logic simulator, but not to a degree of accuracy or generality required in a serious design tool. Researchers at other laboratories have developed their own switch-level simulators based on these earlier designs.

Switch-level simulators can simulate almost the full range of circuit designs available to the MOS designer without any special logic levels or poorly-defined logic elements. Both logic gate and pass transistor combinational logic are simulated without difficulty, as are both static and dynamic memory. A wide variety of bus structures can be simulated including tri-state busses, pre-charged busses, etc. Most significantly, the user need not tell the simulator what type of logic structure is intended, only the actual physical structure of the design.

Switch-level simulators have been tested on a wide variety of MOS designs ranging from student homework problems to a LISP microprocessor chip containing over 10,000 transistors [21]. They have proved remarkably general and accurate, correctly simulating logic design techniques which were not even anticipated in the simulator design. The confidence in the simulation results is greatly enhanced by the fact that the user can see an exact correspondence between the actual design and the simulation network. Moreover, the simulation is fast enough that entire designs can be simulated. For the LISP microprocessor chip, MOSSIM requires between 5 and 12 seconds of CPU time on a DEC20/60 to simulate each clock cycle. The designers were able to fully test the system by simulating 700 clock cycles. Experience has shown that simulation inevitably uncovers fatal errors in the design.

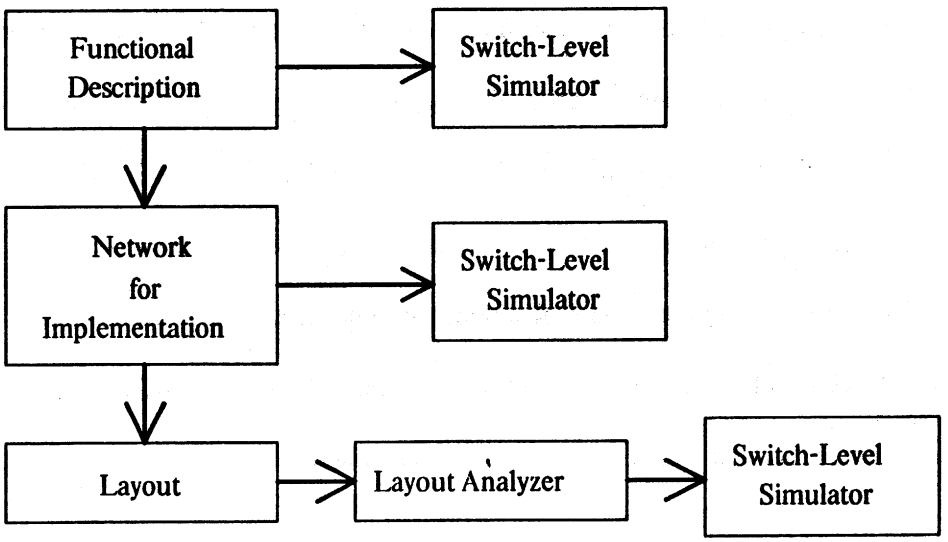
C. Baker [4] has written a program which can take layouts specified in the Caltech Intermediate Form [28] and generate the equivalent transistor-level network. Unlike a program which generates a logic gate description, this program needs no special intuition about logic design. It need only look for electrical connectivity and transistors in the mask patterns. The simulation network for the LISP

microprocessor chip can be generated with 30 minutes of CPU time on a PDP 11/70. This technique has proved extremely valuable, uncovering errors both in the layout and the logic design. Furthermore, it saves the duplicative effort of entering the design by hand for the two different representations. Switch-level simulators can fit into the MOS LSI design cycle at several levels, being applied independently to the high level description, the actual design, and the layout, as shown in Figure 1.4.

1.4 An Abstract Model for MOS LSI

The generality and accuracy of switch-level simulators suggest that a formal model of MOS based on the switch-level concept can be developed. The uniformity and consistency of the switch-level approach are precisely those properties which make a concept amenable to a mathematical treatment. This model would serve not only as a basis for verifying the correctness of a logic simulator, but also as the foundation for new computer tools for MOS design. One need only look at the many advances made possible by Shannon's development of logic models based on Boolean algebra [38, 39] to understand the value of abstract logic models. While the expected benefits of a MOS logic model are much more

Fig. 1.4. Role of Switch-Level Simulation in LSI Design



modest, its utility could be significant. Unlike traditional switching theory which was developed to help humans analyze and synthesize networks containing small numbers of elements, we now want models which can form the basis of computer programs to be applied to networks containing thousands of elements. This places more emphasis on the generality and uniformity of the model and the algorithmic complexity with which it can be implemented.

In this thesis, a formal switch-level model of MOS logic networks is developed. The network model closely resembles the network model of MOSSIM but generalizes it in several respects. As with MOSSIM, a logic network consists of a set of nodes interconnected by a set of transistors. Unlike MOSSIM, however, only two types of nodes: input and normal are allowed. To model ratioed circuits, transistors may have different strengths, with a stronger transistor (such as an inverter pulldown) being able to override a weaker one (such as a pullup load transistor). A third type of transistor, d-type (for "depletion") is introduced which is closed regardless of the gate signal. This new model more closely matches the actual structure of MOS networks, because in MOS networks the relative sizes of transistors determine the logical behavior. The pullup node used in MOSSIM is a rather *ad hoc* way of representing this. The new model can also describe a wider variety of networks, including circuits which rely on multiple levels of ratioing.

In MOSSIM, each normal node is modeled as having a capacitance of unknown value which can store a signal dynamically but cannot drive this signal onto another node in a different state. Unfortunately this model cannot describe the behavior of many bus designs in which a relatively high capacitance bus node is connected to a lower capacitance node (such as the storage node of a 3-transistor dynamic RAM cell) resulting in both nodes obtaining the same logic state as was originally on the bus. Our new switch-level model can model this effect by assigning each normal node a *size*, where the signal on a larger node will predominate when connected to a smaller node.

Our abstract model describes both the time and electrical behavior of a network in a highly idealized way. The time behavior is described by the *target state* function giving the logic states which the normal nodes would reach for a particular set of input node, transistor, and initial normal node states. For designs containing no critical races, the logical behavior can be modeled by repeated application of the target state function. To model the electrical behavior, the target state is defined in terms of the set of steady state voltages in an "order of magnitude" electrical network. This class of networks models the conducting transistors by linear resistors, where the conductances of the resistors for different strength transistors differ by orders of magnitude. As a consequence, any path to an input node containing only transistors with strength greater than or equal to some value is modeled as overriding any path containing a transistor with strength less than this value. Similarly, the normal nodes are modeled by capacitors where the capacitances of the capacitors for different size nodes differ by orders of magnitude. As a result, the target states formed on a set of nodes through charge sharing depends only on the state(s) of the largest node(s) in the set. Furthermore, no attempt is made to accurately compute the node voltages. Instead, they are classified into the three logic states `0`, `1`, and `X`. This model provides a simplified view of ratioed circuits and charge sharing which adequately describes the logical behavior of most MOS circuits.

Although the target state is defined in terms of an electrical model, we will find that the target state of an arbitrary switch-level network can be computed without evaluating any electrical networks. Instead, by introducing an abstraction called *logic signals*, an iterative method for computing the target state can be developed which uses only operations in a simple, discrete algebra. A logic signal provides a composite description of a switch-level network at some node for a particular set of node and transistor states, much as a Thevenin network [15] provides a composite description of a linear network at some port for a particular set of network parameters. However, whereas finding the Thevenin equivalent generally requires solving a set of simultaneous linear equations, finding the logic signal requires much less effort. A simple set of rules describes the logic signals created by the input and normal nodes in their initial states and the effects on a node of other nodes connected through conducting transistors. With these

rules we can develop an equation expressing a set of constraints which must be satisfied by the signal for each normal node in terms of the initial node signal and the signals for input nodes and for other normal nodes connected through conducting transistors. It can then be shown that the minimum solution of this set of equations equals the set of logic signals describing the network at each node, and the set of steady logic states can easily be derived from this solution. Logic signals can be formalized into simple, discrete algebra with a domain corresponding to the signal values and operations describing the effects of the rules for logic signals. This algebra allows us to apply elementary concepts from abstract algebra and lattice theory to develop techniques for computing the target state. These techniques can be further developed into efficient simulation algorithms.

1.5 Relation to Relay Networks

The switch-level MOS model can be viewed as an extension of Shannon's relay network model [38, 39]. The algebra of signal strengths introduced in Chapter 5 bears many similarities to Boolean algebra. As Shannon observed, a relay can be viewed as a switch with conductance 1 when closed and 0 when open.[1] The rules for connecting relays in series and in parallel and the methods of analyzing relay networks are special cases of those for transistor networks.

MOS networks, however, have several characteristics which are not found in relay networks. First, relay networks are used as a *current-driven* logic, in which the logic state of a node is determined by the connection between the node and the current source. Thus a simple characterization of the connection to the signal source determines the state of a node. MOS networks, in contrast, are used as a *voltage-driven* logic, where the state of a node is determined by both its connection to the supply voltage and its connection to ground. One must characterize both the states of the signal sources and the connections to them to determine the state of a node. Furthermore, in a voltage-driven logic erroneous behavior may

---

1. Shannon actually described the state of a relay by its *hindrance*, the complement of its conductance.

result due to a short circuit between signal sources, and hence we also require the state X for logical completeness. Second, relay networks do not allow ratioing in which one closed switch can override another. Thus, a Boolean characterization of a conductance path suffices. Third, relay networks can only store information in feedback paths. No modeling of dynamic memory or charge sharing is needed. Finally, most theoretical work on relay networks was conducted before the widespread availability of digital computers. Thus, most techniques were developed to aid the hand design of small circuits. The standards by which ideas are measured change greatly when they are to be incorporated into computerized tools to aid the design of very large systems.

1.6 Outline of Thesis

In the next chapter, the details of how the switch-level model describes the structure and operation of MOS networks is presented. By modeling the network structure at a transistor level and by allowing transistors of different strengths and nodes of different sizes, this model covers a large variety of MOS design techniques in a way which closely matches the actual circuit designs. The switch-level model can be viewed as either a simplification of analog network models or an extension of relay network models. The time behavior of a network is described by the target state function, giving the logic states toward which the nodes are driven or charged given the current node and transistor states. The value of this function is defined in terms of the steady state voltages in a linear electrical network that models the transistor network. The logical behavior of many MOS networks is described by repeated applications of their target state functions. In computing the target state, the transistors are modeled with time-invariant elements, thereby simplifying the analysis considerably.

In Chapters 3 and 4 the electrical circuit-oriented view of the target state function provided by the definition given in Chapter 2 is transformed into a more abstract and logical view. It is shown that the target state can be defined in terms of the *steady states* of a set of *logical conductance* networks, where a logical conductance network represents a switch-level network in which each transistor is either

nonconducting or fully conducting. The concept of logic signals is then developed to express the behavior of logical conductance networks. With the logic signal abstraction we can derive an equation which gives the steady state of a logical conductance network, and consequently the target state of a switch-level network, which does not require evaluating any electrical networks.

In Chapter 5, an algebra of logic signals is developed with operations describing the effects of a set of network transformations. This algebra allows us to apply elementary concepts from abstract algebra and lattice theory to the study of switch-level networks.

In Chapter 6 the mathematical formalism presented in Chapter 5 is used to derive a technique for computing the target state of a switch-level network. This development utilizes only the logic signal abstraction as expressed by the algebra of Chapter 5 and two equations which are derived in Chapters 3 and 4 from the analysis of the electrical model. Although we could arrive at the desired results more directly by utilizing additional properties of the electrical model, this approach demonstrates the power of our abstract approach.

In Chapter 7, the abstract solution technique of Chapter 6 is developed into an efficient algorithm for a switch-level simulator. By exploiting the sparseness of the network, the simulator requires at most linear time to simulate one clock cycle for almost all networks. This algorithm improves on previous switch-level simulation algorithms in several respects. Some performance data for MOSSIM is presented to demonstrate the performance characteristics of switch-level simulation and how it compares to logic gate simulation.

In Chapter 8, the simplified timing model of MOSSIM is investigated more closely to see for what classes of systems it is valid. Possible methods of implementing logic simulators with other timing models are presented. In addition, a *ternary* simulation algorithm is developed which uses the X state to detect potential races in MOS networks. This algorithm is a straightforward extension of Brzozowski and Yoeli's algorithm for logic gate networks [11]. Ternary simulation requires a much more accurate and efficient implementation of the X state than is required for functional simulation, because the X state will become

the most prevalent state in the network. The algorithm presented in Chapter 7 provides this accuracy and efficiency.

Finally, in Chapter 9 some ideas for further improvements of logic simulators and for future applications of the switch-level model are described.

1.7 Notation

In the remainder of this presentation, the following notational conventions are observed. Scalar values are denoted with lower case letters (e.g. a, b); vectors with boldface, lower case letters (e.g. **a**, **b**); and matrices with boldface, upper case letters (e.g. **A**, **B**). Mathematical domains, i.e. sets of values with particular mathematical properties are denoted with script, upper case letters (e.g. $\mathcal{A}$, $\mathcal{B}$), while ordinary sets are written with italic, upper case letters (e.g. *A*, *B*). The extension of a domain $\mathcal{D}$ to vectors of size n is denoted $\mathcal{D}^n$, and the extension to matrices with n rows and m columns is denoted $\mathcal{D}^{n \times m}$.

2. The Switch-Level Network Model

2.1 Introduction

In this chapter a model of the logical structure and operation of MOS networks will be presented. This model attempts to capture those aspects of an MOS circuit which affect its logical behavior while ignoring many of the detailed electrical properties. This network model extends the network model of MOSSIM but in a way which provides a consistent level of abstraction. Like MOSSIM the network can be extracted directly from a specification of the layout. The time behavior of the network is also described in a simplified way by the *target state* function. Given the current network state, this function yields the state toward which the nodes move without considering the rate at which these changes occur. This function bears a strong resemblance to the *excitation* function used in logic gate and relay models. It is assumed that the reader has a background in MOS logic design comparable to that provided by Mead and Conway [28].

2.2 Network Structure

A logic network contains a set of input nodes $I = \{ i_1, \ldots, i_m \}$, a set of normal nodes $N = \{ n_1, \ldots, n_n \}$, and a set of transistors $T = \{ t_1, \ldots, t_k \}$.

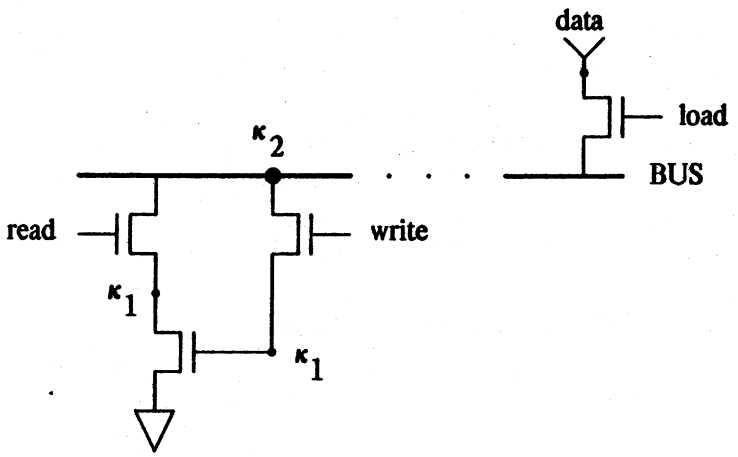
Input nodes provide strong signals from sources external to the network, much like voltage sources in electrical networks. Examples of input nodes include the supply (VDD) and ground (GND) connections as well as signals supplied through input pads. *Normal* nodes have states determined by the operation of the network. Each normal node n_i has a *size* cap_i, where cap_i is an element in the set $K = \{ \kappa_1, \ldots, \kappa_q \}$. A normal node can store charge to provide dynamic memory. The size of a node gives an approximate characterization of the amount of charge it can store, where sizes are ordered

$$\kappa_1 < \kappa_2 < \ldots < \kappa_q$$

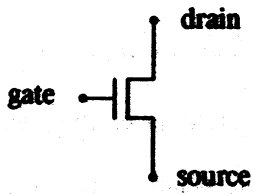
When normal nodes are connected together they will share charge and settle in a state dependent only on the state(s) of the largest node(s). The values in K have no properties other than their ordering; they only indirectly represent actual physical capacitances. This model provides a simplified view of charge sharing which is valid for most actual circuit designs. The number of node sizes q depends on the kinds of MOS networks to be modeled. For most networks, $q = 1$ will suffice. For those networks which rely on a sharing of charge between a high capacitance node and a low capacitance node for their logical behavior, q must equal 2 or more. For example, Figure 2.1 shows a three-transistor dynamic RAM circuit which relies on the high capacitance of the bus (size κ_2) to override the charge on the storage node of the cell during a write and on the drain node of the storage transistor during a read.

Fig. 2.1. Ratioless MOS Design

Three Transistor Dynamic RAM



A transistor is a three terminal device as shown below.



No distinction is made between the source and drain connections. Associated with each transistor t_i is a *strength* str_i, where str_i is in the set $\Gamma = \{ \gamma_1, \ldots, \gamma_p \}$. The strength of a transistor gives an approximate characterization of its conductance when turned-on, with strength values ordered

$$0 < \gamma_1 < \gamma_2 < \ldots < \gamma_p$$

The strength values in Γ have no properties other than their ordering; they only indirectly represent actual conductances. The number of allowable strengths p depends on the kinds of networks to be modeled. For networks which do not rely on ratioed resistances such as CMOS designs, all transistors are of equal strength and $p = 1$. Most ratioed nMOS or pMOS designs can be modeled with $p = 2$, where pullup and pulldown loads have strength γ_1 and all other transistors have strength γ_2. Some designs, including certain static RAM cells rely on multiple levels of ratioing and hence require a model with p equal to 3 or more. A transistor can be either n-type, p-type, or d-type. All act as voltage-controlled switches as follows:

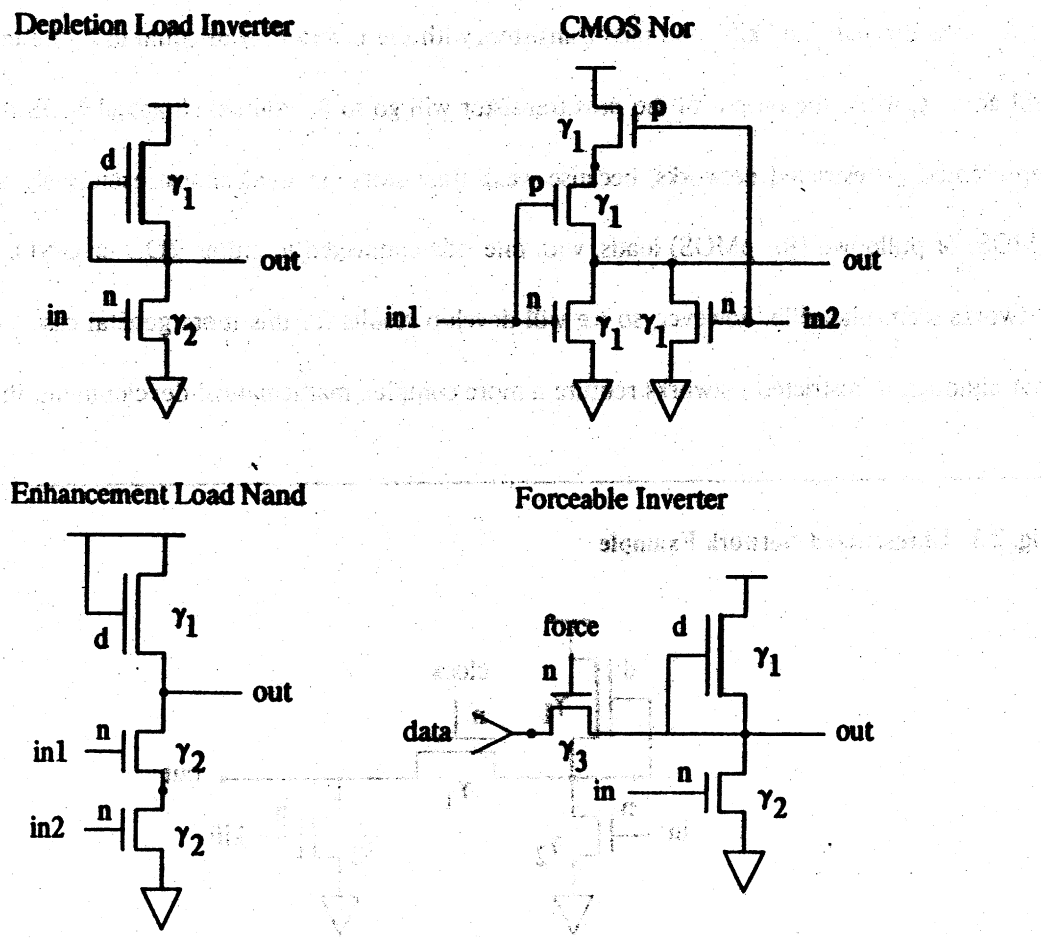
n-type		p-type		d-type	
gate signal	effect	gate signal	effect	gate signal	effect
0	open	0	closed	0	closed
1	closed	1	open	1	closed
X	unknown	X	unknown	X	closed

A d-type (for "depletion") transistor can serve as either a load resistor for a depletion mode nMOS logic gate, or a polysilicon-diffusion crossover such as is seen in some designs. When a transistor is in a "closed" state it provides a conductance between the source and drain nodes with value characterized by the transistor strength. When a transistor is in an "unknown" state it provides a conductance of unknown

value between (inclusively) the conductance when "open" (i.e. 0.0) and when "closed". This model provides a simplified view of ratioed circuits in which a connection through a stronger transistor will always override a connection through a weaker, as will be defined more rigorously later in this chapter.

Examples of a variety of MOS circuits are shown in Figure 2.2. The first three show common nMOS and CMOS logic gates. The fourth one shows a forceable inverter which, when connected in a ring with another inverter, can statically store 1 bit.

Fig. 2.2. Examples of MOS Networks

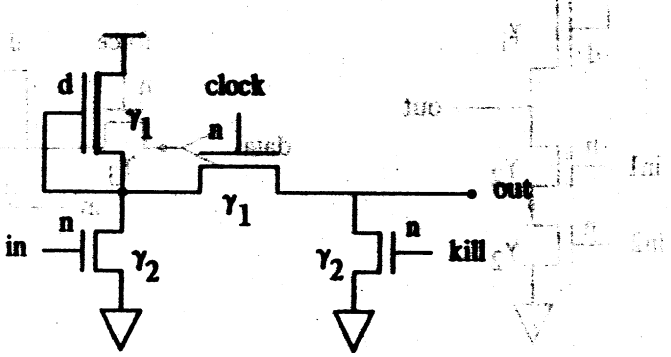


Transistors with strength less than γ_p (called weak transistors) are used in ratioed circuits, where a stronger transistor may override a weaker one. Generally these weak transistors are configured only in limited ways, such as pullup loads on logic gate outputs. At times we will want to exploit the characteristics of networks restricted in the use of weak transistors to simplify the mathematical development or to improve the efficiency of an algorithm. In particular, we define a *restricted* network follows:

> In a restricted network every transistor of strength less than γ_p has either its source or drain connected to an input node.

All of the circuits in Figure 2.2 are restricted networks. An example of an unrestricted network is shown in Figure 2.3. In this example the pass transistor is assigned a strength γ_1 to indicate that when a sneak path forms through the "kill" and pass transistors with the inverter input equal to 0, the inverter output will go to X, while the output of the pass transistor will go to 0. Almost all actual MOS designs can be represented as restricted networks, because weak transistors are used almost exclusively as pullup (for nMOS) or pulldown (for pMOS) loads, with one side connected to either VDD or GND. Unrestricted networks seem plausible, however, so we will develop results for this more general case. We shall find that although unrestricted networks require a more complex mathematical development, their study will

Fig. 2.3. Unrestricted Network Example



provide much insight into networks containing transistors with gate nodes in the X state.

2.3 Network Representation

The structure of a network is represented by the sets *I*, *N*, and *T*, the vectors cap and str (indicating the node sizes and transistor strengths), and the following functions:

TTYPE:	$T \rightarrow \{ n, p, d \}$	the transistor type
GATE:	$T \rightarrow I \cup N$	the gate node
SOURCE:	$T \rightarrow I \cup N$	the source node
DRAIN:	$T \rightarrow I \cup N$	the drain node

2.4 Logic States

Each normal node n_i has a logic state $y_i \in \{ 0, 1, X \}$. In terms of the actual voltage v_i at node n_i:

$$y_i = 0 \quad \Rightarrow \quad 0.0 \leq v_i \leq V^-$$

$$y_i = 1 \quad \Rightarrow \quad V^+ \leq v_i \leq V_{dd}$$

$$y_i = X \quad \Rightarrow \quad 0.0 \leq v_i \leq V_{dd}$$

where V^- and V^+ are the logic thresholds. Note that our logic model makes no attempt to accurately model these logic thresholds, and they are used here only to aid our informal discussion. Each input node i_i has a logic state $x_i \in \{ 0, 1, X \}$ with the same interpretation. The values 0 and 1 correspond to the Boolean logic levels. The value X indicates either an *unknown* or *undefined* logic level. An *unknown* logic level arises when an ambiguity in the network condition prevents a unique determination of a node's logic level, such as from uninitialized state variables. For example, when power is applied to a bistable device such as a Nor gate latch, the output will obtain a valid, but unknown logic state. An unknown level corresponds to a voltage either below V^- or above V^+. An *undefined* logic level arises when the network operation creates a voltage which could lie between the two logic thresholds, such as due to a short circuit or improper charge sharing. Hence an undefined level corresponds to a voltage which may be anywhere between 0.0 and V_{dd}.

These two concepts differ slightly in that an "unknown" value satisfies the Law of Excluded Middle, while an "undefined" value does not. For example, if y represents the state of an uninitialized bistable device, then $y + \neg y = 1$. On the other hand, if y represents the state of a node along a shorting path from VDD to GND, then $y + \neg y = X$. Some circuit designs exploit the Law of Excluded Middle during the initial power-up sequence to assure that all feedback paths will be initialized to valid logic levels. No known algorithm, however, can utilize information about "unknown" logic values in a completely general way, except by enumerating over all possible combinations of Boolean values. Thus, to avoid an exponential algorithm, we shall not attempt to distinguish between the "unknown" and "undefined" logic levels but instead use the single value X.

Each transistor t_j also has a logic state $z_j \in \{ 0, 1, X \}$, where 0 indicates "open", 1 indicates "closed", and X indicates "unknown". Although transistor states and node states are different physical phenomena, we will use the same mathematical objects to represent both.

2.5 Network State

At any instant in time the state of a network is given by the the logic states of the input nodes $x \in \{ 0, 1, X \}^m$, the states of the normal nodes $y \in \{ 0, 1, X \}^n$, and the states of the transistors $z \in \{ 0, 1, X \}^k$. Under stable conditions, the transistor states z are functions of the node states. Suppose, for example, that node n_i is the gate node for transistor t_j (i.e. $n_i = \text{GATE}(t_j)$). Then the value of z_j is given by the following table

	TTYPE(t_j)		
y_i	n	p	d
0	0	1	1
1	1	0	1
X	X	X	1

A similar table would result if the gate node were an input node. The function *trans*(x, y) denotes the transistor state z resulting from the node states x and y. During the actual circuit operation, some time

may elapse between a change in node state and the resulting change in transistor state. Hence the total state of the network must include the transistor states as well as the node states.

2.6 The Target State Function

With the network in state (x, y, z) each normal node n_i will move toward a *target state* $\tilde{y}_i$ given by the function

$$\tilde{y}_i = target_i(x, y, z).$$

The target state of a node equals the state the node would eventually reach if the input nodes were held fixed in state x, the transistors were held fixed in state z, and the normal nodes were initialized to state y. As shall be seen, the target state $\tilde{y}$ is the steady state solution of a time invariant network with parameters x and z and initial conditions y. We can view *target* as a vector-valued function

$$\tilde{y} = target(x\ y, z). \tag{2.1}$$

giving the target states for all normal nodes. The state $\tilde{y}$ may never actually be reached, however, because the transistor states will change in response to the changing node states, and the input node states may be changed externally. The function *target* only describes a tendency in the network and not a definite reality. However, it provides a basic characterization of the logical behavior of a switch-level network. Much of the development of this thesis will be directed toward a mathematical formulation of the target state function.

Define the function $step_x$ as

$$step_x(y) = target(x, y, trans(x, y)). \tag{2.2}$$

For a particular state of the input nodes $step_x$ gives the target states of the normal nodes as a function of their initial states, assuming the transistor states are functions of the initial node states.

During actual operation, the network may not move through the succession of states predicted by the function $step_x$ due to changing transistor states and changing input conditions. However, for a large class of networks, the ultimate behavior is equivalent to the behavior modeled by successive applications of the function $step_x$ as long as the input nodes remain unchanged. That is, if the normal nodes initially have state y and the input nodes are held fixed in state x, the network will eventually reach a state *phase*(x, y) defined as

$$phase(x, y) = \lim_{k \to \infty} step_x^k(y) \tag{2.3}$$

where the superscript k indicates k applications of the function $step_x$. Furthermore, the networks of interest will be guaranteed to stabilize after a bounded number of steps. Thus for some $k < \infty$, $step_x^k(y) = step_x^{k+1}(y) = phase(x, y)$. Once the network arrives at this state, it will remain there until some input node is changed. This ignores the possibility that nodes may lose their charge due to leakage. With current technology, in which clock speeds are measured in megahertz while leakage times are measured in milliseconds, this assumption is appropriate. Examples of systems which can be modeled by repeated applications of $step_x$ include combinational logic, any system free of critical races, and a variety of systems which can be modeled with unit delay logic elements. The limitations of this assumption will be discussed in Chapter 8.

For a large class of MOS systems an equation for the function *target* will lead to a description of the complete functional behavior of a network. In other words, the behavior of a system can be modeled by repeatedly freezing the states of the nodes and transistors, computing the target state for each normal node, and then updating the node and transistor states accordingly. This technique has obvious advantages over modeling the detailed voltage waveforms on each node.

Concepts similar to the target state function have been applied to the study of both relay networks and logic gate networks. The target state function describes the *excitation* of a switch-level network, i.e. the node states created in response to the current states. Similarly, the excitation of a relay network is defined as the states which would appear on the relay coils if all relay contacts are held fixed in their current states, while the excitation of a logic gate network is defined as the outputs of the logic gates as functions of their current inputs. In general, the excitation of any logic network can be defined as the logic states which would form on the nodes if all active elements (e.g. transistors, relays, or logic gates) were held fixed in their current states.

Huffman [22] first recognized the importance of the network excitation as providing a basic characterization of the dynamic behavior of a logic network, although he expressed it in the form of a flow table rather than a function. With this characterization, much of the physical behavior is abstracted away, such as the rate at which the nodes move toward through their excitations and the analog values through which they pass. While such a characterization cannot detect certain error conditions dependent on the detailed voltages or timing, it provides a useful level of abstraction. Although logic states are formed in switch-level networks in much different ways than in logic gate or relay networks, these three kinds of networks share much in common when viewed as systems computing logical functions.

2.7 Specification of the Target State

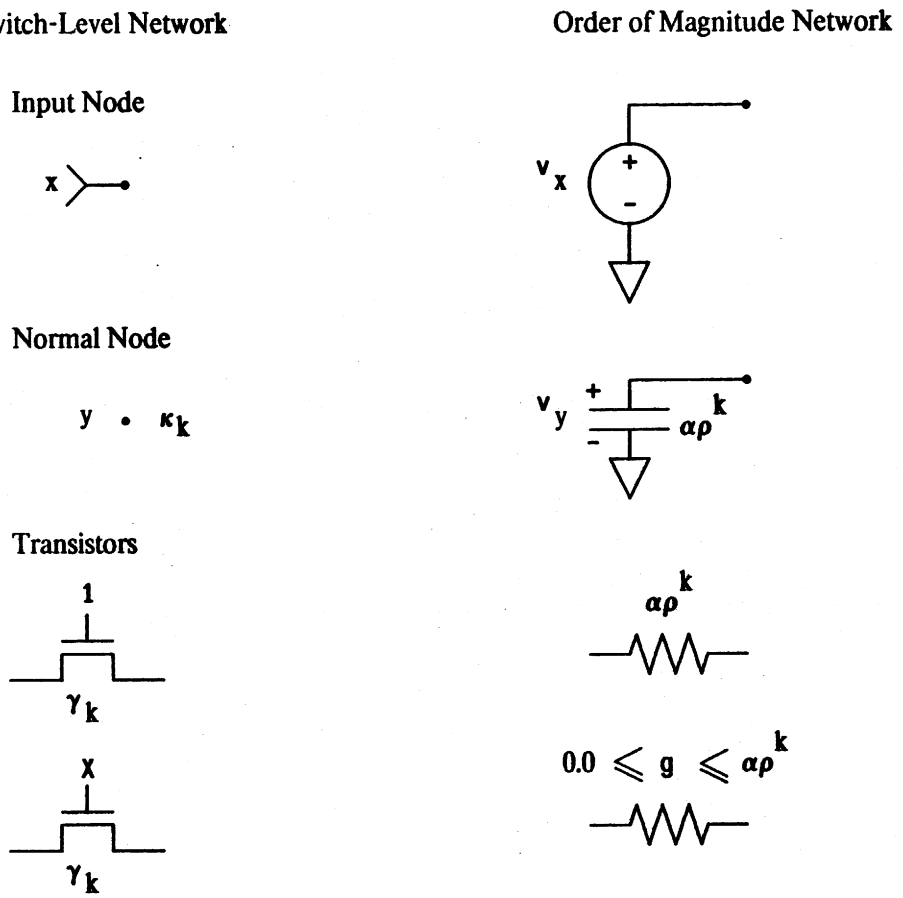
For both relay and logic gate networks, the excitation function for a node can be defined with a set of Boolean functions in a relatively straightforward way. With switch-level networks, on the other hand, a more complex formulation is required, because the logic elements are bidirectional, and because the state depends on the relative conductances of the pullup and pulldown paths acting on it or on the relative capacitances of nodes with which it may share charge. We will define the target state in terms of a linear electrical model called the "order of magnitude" model. With this model the concepts behind the switch-level model can be expressed in relatively conventional mathematical terms. In later chapters, a

rather unconventional algebra is presented to express these concepts more directly and to allow the development of efficient simulation algorithms. The order of magnitude electrical model serves only as a means by which these more abstract concepts can be motivated and derived.

Order of magnitude networks contain voltage sources, resistors, and capacitors where the resistor conductances and capacitor capacitances are specified as powers of a ratio parameter ρ. Transistors in the X state form conductances of unknown value bounded by powers of ρ, and nodes in the X state form unknown voltages. Hence, we must consider the set of possible steady state voltages for each node in an order of magnitude network when the parameters and initial conditions range over sets of values. The target state of a node in a switch-level network is defined in terms of the set of possible steady state voltages on the corresponding node in the order of magnitude network as ρ is made very large, such that the conductances of different strength transistors in the 1 state and the capacitances of different size nodes differ by orders of magnitude. This provides a simplified view of ratioed logic and charge sharing in which only the dominating effects are considered.

The order of magnitude network corresponding to a switch-level network in a particular state (x, y, z) is constructed with elements shown in Figure 2.4. Each input node i_j is modeled by a voltage source with negative terminal connected to GND and with voltage x_j, where $x_j = 0.0$ if $x_j = 0$, $x_j = V_{dd}$ if $x_j = 1$, and x_j ranges over the set of voltages $\{ v \mid 0.0 \leq v \leq V_{dd} \}$ if $x_j = X$. A normal node n_j of size κ_k is modeled by a capacitor with one side connected to GND and with capacitance $\alpha\rho^k$ where α is an arbitrary positive constant. This capacitor is initially charged to a voltage y_j defined in terms of the logic state y_j in the same way x_j is defined in terms of x_j. A transistor with strength γ_k and state 1 is modeled by a resistor of conductance $\alpha\rho^k$ where α is an arbitrary positive constant. A transistor with strength γ_k and state X is modeled by a resistor with conductance ranging over the set $\{ g \mid 0.0 \leq g \leq \alpha\rho^k \}$, where α is the same constant used when the transistor state is 1. The values of α can be different for different resistors and capacitors. It will be shown later that the values of these coefficients do not affect the value of the target state.

Fig. 2.4. The Order of Magnitude Network Model



For a particular value of ρ, the conductance parameters in an order of magnitude network can be described by two matrices $\mathbf{G} \in \mathfrak{R}^{n \times n}$ and $\mathbf{E} \in \mathfrak{R}^{n \times m}$, where $\mathfrak{R}$ denotes the set of real numbers. Each element g_{ij} of $\mathbf{G}$ equals the sum of all conductances between nodes n_i and n_j, while each element e_{ij} of $\mathbf{E}$ equals the sum of all conductances between node n_i and the voltage source corresponding to input node i_j. When the switch-level network contains transistors in the X state, these matrices may range over infinite sets:

$$\mathbf{G}^{min}(\rho) \leq \mathbf{G} \leq \mathbf{G}^{max}(\rho)$$
$$\mathbf{E}^{min}(\rho) \leq \mathbf{E} \leq \mathbf{E}^{max}(\rho)$$

with the restriction that $\mathbf{G}$ be symmetric. Note that the partial order $\leq$ is defined between matrices in the

usual way, i.e. $\mathbf{A} \leq \mathbf{B}$ if and only if $a_{ij} \leq b_{ij}$ for all i and j. The elements of $\mathbf{G}^{min}$, $\mathbf{G}^{max}$, $\mathbf{E}^{min}$, and $\mathbf{E}^{max}$ are polynomial functions of ρ, and hence the elements of the conductance matrices $\mathbf{G}$ and $\mathbf{E}$ are bounded by polynomial functions of ρ, although they may take on arbitrary values within these ranges. With this formulation, we are assuming that transistors with the same gate node behave independently when that node is in the X state. This models the possibility that transistors may have slightly different threshold voltages, and hence when the gate node has a voltage close to one of the thresholds, the transistors may behave quite differently. Furthermore, as shall be shown in Chapter 3, this assumption allows us to look only at the minimum and maximum conductance values for each transistor when computing the target state.

The remaining parameters of an order of magnitude network are described by a vector $\mathbf{c}(\rho) \in \mathfrak{R}^n$, giving the capacitances corresponding to the normal nodes, and the vector $\mathbf{x} \in \{ v \mid 0.0 \leq v \leq V_{dd} \}^m$ giving the settings of the voltage sources corresponding to the input nodes. When the switch-level network contains input nodes in the X state, this vector may range over an infinite set

$$\mathbf{x}^{min} \leq \mathbf{x} \leq \mathbf{x}^{max},$$

where if $x_i = X$, $x^{min}_i = 0.0$ and $x^{max}_i = V_{dd}$. The initial conditions of the order of magnitude network are described by the vector $\mathbf{y} \in \{ v \mid 0.0 \leq v \leq V_{dd} \}^n$ giving the initial voltages on the capacitors corresponding to the normal nodes. When the switch-level network contains normal nodes initially in state X, this vector may range over an infinite set

$$\mathbf{y}^{min} \leq \mathbf{y} \leq \mathbf{y}^{max}.$$

Let $v_i(\mathbf{G}, \mathbf{E}, \mathbf{c}(\rho), \mathbf{x}, \mathbf{y})$ denote the steady state voltage on node n_i for the network with parameters $\mathbf{G}$, $\mathbf{E}$, $\mathbf{c}(\rho)$, and $\mathbf{x}$, and initial conditions $\mathbf{y}$. Since the network contains only passive, linear elements, this voltage must be unique. Furthermore, since the network contains no floating capacitors, all node voltages must lie between 0.0 and V_{dd}. When nodes or transistors in the X state are present in a switch-level

network, this voltage can range over a set $V_i(\rho)$ where

$$V_i(\rho) = \{ v_i(G, E, c(\rho), x, y) \mid G = G^T, G^{min}(\rho) \leq G \leq G^{max}(\rho), E^{min}(\rho) \leq E \leq E^{max}(\rho), x^{min} \leq x \leq x^{max}, y^{min} \leq y \leq y^{max} \}.$$

This set is uniquely determined by the structure and state of the switch-level network, the constant coefficients α (which will be seen to be unimportant), and the ratio parameter ρ.

The target state of a node n_i is defined in terms of the set $V_i(\rho)$ as the ratio parameter ρ is made very large. As this occurs, any conductance paths to input nodes formed by transistors with state 1 and strength greater than or equal to γ_k will dominate over paths containing transistors of strength less than γ_k. Similarly, the charge on capacitances formed by normal nodes of size κ_k will dominate over the charge on capacitances formed by normal nodes of lesser size. To form a proper logic state (i.e. 0 or 1), there can be no conflict between the pullup and pulldown paths or between the high and low charges. As ρ is made very large $V_i(\rho)$ should approach either the set $\{ 0.0 \}$ or the set $\{ V_{dd} \}$. If we take the limit as ρ approaches infinity, the set $V_i(\rho)$ should converge to one of these two sets. Thus, if we define the set V_i^{∞} as

$$V_i^{\infty} = \lim_{\rho \to \infty} V_i(\rho),$$

then the target state on node n_i is defined as

$$\tilde{y}_i = \begin{cases} 0, & V_i^{\infty} = \{ 0.0 \} \\ 1, & V_i^{\infty} = \{ V_{dd} \} \\ X, & \text{else.} \end{cases} \qquad (2.4)$$

In this formulation, the X state arises either when the set $V_i(\rho)$ converges to some value not equal to 0.0 or V_{dd}, indicating erroneous behavior due to a short circuit or improper charge sharing, or when the set $V_i(\rho)$ fails to converge to a single value, indicating an ambiguity in the steady state voltage caused by nodes or transistors in the X state. This definition of the target state in terms of a limiting process expresses in a mathematical way the concept that the switch-level model considers only the dominating

effects acting on each node, either through conductance paths formed by transistors in the 1 state to input nodes or by charge sharing between normal nodes. When the dominating effects conflict or when nodes or transistors in the X state create uncertainties, the target state equals X.

For example, Figure 2.5 shows a switch-level model of an nMOS Nor gate and the corresponding order of magnitude network. Let us see how the target states would be defined for several sets of inputs. If $in_1 = 1$ and $in_2 = X$, then $g_1 = \alpha_1\rho$, $g_2 = \alpha_2\rho^2$, and $0.0 \leq g_3 \leq \alpha_3\rho^2$, where α_1, α_2, and α_3 are arbitrary positive constants. This gives a set of steady state voltages for node n_i

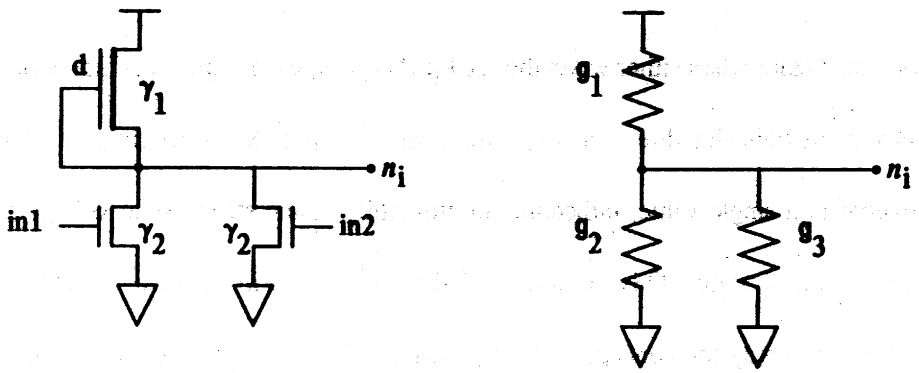
$$V_i(\rho) = \left\{ v \;\middle|\; \frac{\alpha_1 V_{dd}}{\alpha_1 + (\alpha_2 + \alpha_3)\rho} \leq v \leq \frac{\alpha_1 V_{dd}}{\alpha_1 + \alpha_2\rho} \right\},$$

and therefore $V_i^{\infty} = \{ 0.0 \}$ and $\tilde{y}_i = 0$. Now suppose that $in_1 = 0$ and $in_2 = X$. Then $g_1 = \alpha_1\rho$, $g_2 = 0.0$, and $0.0 \leq g_3 \leq \alpha_3\rho^2$. This gives a set

$$V_i(\rho) = \left\{ v \;\middle|\; \frac{\alpha_1 V_{dd}}{\alpha_1 + \alpha_3\rho} \leq v \leq \frac{\alpha_1 V_{dd}}{\alpha_1} \right\},$$

and therefore $V_i^{\infty} = \{ v \mid 0.0 \leq v \leq V_{dd} \}$ and $\tilde{y}_i = X$.

Fig. 2.5. Example of an Order of Magnitude Network



2.8 Relation to Actual Circuits

We have defined the target state in terms of an electrical network model as the ratio between the conductances of different strength transistors and between the capacitances of different size nodes is made very large. It then follows that the switch-level model would correctly describe an MOS circuit containing only transistors in which the length-width ratios differ by orders of magnitude along with nodes with capacitances that also differ by orders of magnitude.

In designing actual MOS circuits, of course, one does not use transistors with length-width ratios which differ by orders of magnitude. Instead, the relative sizes of transistors along a conducting path from VDD to GND are set so that the node voltages will lie sufficiently within the logic thresholds. Furthermore, transistors are sized according to their required speed, power, and driving capabilities. As a result, the pullup load in a clock driver may have much greater conductance than the pullup in an ordinary inverter, even though both perform the same logical function -- to provide a 1 signal in the absence of a stronger 0 signal. This transistor sizing can be viewed as an optimization of our order of magnitude model to improve the electrical characteristics while retaining the same logical behavior. In almost all MOS circuits the logic value formed on a node depends only on the dominating effects.

Similarly, node capacitances span a wide range of values, but the logical behavior is affected by node sizes only in a few isolated locations, such as in pre-charged bus circuits. Typically the large capacitance node (e.g. the bus) greatly exceeds the smaller capacitance node, and hence our order of magnitude model more nearly approximates the actual circuit in this case.

To model the logical behavior of a correctly designed MOS circuit we need only characterize transistor and node sizes according to their logical function in the network. This correctness can be tested prior to the simulation by a computer program which compares the conductance and capacitance parameters of the circuit against the proposed logic network. Thus the simulation model can assume that the circuit is correctly designed in this respect and take a more abstract view of ratioed circuits and charge

sharing.

Not all MOS digital circuits can be modeled by a switch-level network. For example, the switch-level model cannot describe circuits in which slight variations in voltages can represent different logic values, such as one-transistor dynamic RAM designs using sense amplifiers to detect these variations. Furthermore, our model can only describe the behavior of ratioed circuits or charge sharing when there is a clear precedence between the different transistor conductances and between the different node capacitances. Other forms of MOS circuits can only be modeled with partial accuracy. For example, the switch-level model ignores the effects of floating capacitors in "bootstrapping" node voltages above V_{dd} or below 0.0. This technique, however, is used primarily to overcome the saturation effects of the field-effect transistor, but our model ignores these effects anyhow. Thus a circuit which utilizes bootstrapping for this purpose can be simulated with a switch-level model, but the simulation will not check whether the bootstrapping actually occurs. Except for these limitations, switch-level networks provide an accurate and simple way to describe the behavior of MOS logic circuits.

2.9 Comparison to Other Switch-Level Models

The logic networks allowed by the simulator MOSSIM [8] can be described in the model using only one node size ($q = 1$) and two transistor strengths ($p = 2$). Input and normal nodes in MOSSIM networks are modeled as input and normal nodes, while pullup nodes are modeled by one of the following circuits:

The new model corresponds more closely to the actual circuit implementation, because transistors of

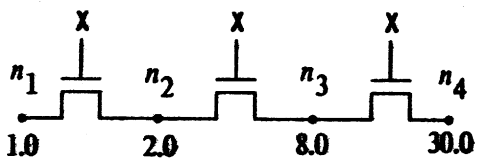
different sizes define the behavior of ratioed circuits. The *pullup* node of MOSSIM provides a rather inelegant model of this and also lacks generality. All transistors in a MOSSIM network are modeled by transistors of strength γ_2. A MOSSIM network always corresponds to a restricted network in the new model.

The logic networks allowed by C. Terman's switch-level simulator [5] are for the most part identical to MOSSIM networks. A more general model of charge sharing is allowed, however, in which each normal node has a real-valued capacitance. This technique is applied when a set of normal nodes is interconnected only by transistors in the 1 state, none are connected to input nodes, and none are in the X state. In such a case, the program sums the capacitances of those nodes in state 1 as well as the capacitances of nodes in state 0. If one sum exceeds the other by at least a factor of 3, the nodes are all set to the corresponding state, and otherwise they are set to X. No attempt is made to perform this calculation when transistors or nodes in the X state are present. Instead, the nodes are all set to X.

At first this approach might seem superior to our model of charge sharing in which nodes are assigned a size in the set K and the value on a larger node always overrides the value on a smaller. Using real-valued capacitances more closely matches the actual electrical behavior and utilizes only information easily calculated from the layout specification. However, it seems as if transistors in the X state cannot be dealt with in a consistent way with this model. Terman has chosen not to simulate the effects of transistors in the X state with great accuracy. Instead, nodes are sometimes set to X even when they would have state 0 or 1 regardless of the conductances of transistors in the X state. For example, if a high capacitance node in state 0 is connected to a low capacitance node in state 1 by a transistor in state X, both nodes are set to X even though the high capacitance node would remain in state 0 even if it shared charge with the other node.

Suppose that we were to utilize real-valued capacitances in our logic model but tried to provide a more accurate model of transistors in the X state. That is, a node would be set to 0 or 1 if it has this unique state regardless of the conductances of transistors in state X, and otherwise it would be set to X. Consider, for example, the network shown in Figure 2.7 containing a set of nodes of increasing capacitance connected by transistors in the X state. Suppose initially that node n_1 is set to 1 and all others are set to 0. With this model, the target states of nodes n_1 and n_2 equal X, because these nodes would have undefined states if only they share charge. Nodes n_3 and n_4, on the other hand, would have target states 0, because no setting of the transistor conductances could cause them to be charged above the logic threshold. We have defined the target state as the steady state solution of the network, and hence when the nodes are set to their target states, the network should remain stable until a transistor or input node changes state. if we set nodes n_1 and n_2 to X, however, the target state of node n_3 will become X, because by our naive approach, we must consider the case in which the X states on nodes n_1 and n_2 actually represent high voltages, and these nodes share charge with just n_3. Hence, the original target state is not a true steady state solution. Similarly, if we set node n_3 to its new target state, the target state of node n_4 becomes X, indicating that the previous target state was also not stable. In the final steady state solution, the initial charge on node n_1 has created an undefined state on a node with 30 times greater capacitance. Thus, this model of charge sharing yields unstable solutions and also lacks accuracy.

Fig. 2.6. Charge Sharing Anomaly



	n_1	n_2	n_3	n_4
initial	1	0	0	0
step 1	X	X	0	0
step 2	X	X	X	0
step 3	X	X	X	X

More sophisticated approaches could be devised such as storing the range of possible voltages for each node, but for any scheme there seems always to be a circuit which would be modeled incorrectly. The problem occurs because we are trying to combine exact electrical concepts such as real-valued capacitance with abstract logical concepts such as the X state for both nodes and transistors. With the logic model mapping many possible conditions of the electrical network into the state X, it cannot model behavior which depends on detailed electrical properties with sufficient accuracy or consistency. By adopting a more absolute view of charge sharing in which a larger node can always override a smaller, we obtain a more uniform level of abstraction leading to a more consistent modeling. Of course to describe a design in terms of our logical model the process translating the electrical design into the logic model must decide how the design should be viewed logically. For the network shown in Figure 2.7 this would require some rather unsatisfying decisions. In fact this network really should be modeled as an analog circuit, because its behavior depends too much on exact electrical properties.

This aspect of the logic model design indicates a fundamental trade-off with abstractness and consistency on one hand and a desire to combine concepts from several different models on the other. Since we are concerned at the moment with developing the mathematical aspects of the switch-level model, the more consistent and abstract approach will be chosen. For other applications, a different choice may be appropriate.

2.10 Derivation of the Switch-Level Network

Switch-level simulators have proved quite successful in simulating networks extracted by a computer program directly from a description of the mask layouts. The combination of network extraction and simulation provides an important check of a design.

For networks which can be expressed in the MOSSIM network model, this layout extraction is relatively straightforward. The program need only follow the electrical connectivity in the network and find the transistors and their types. A simple set of rules allow one to translate this node and transistor description into a MOSSIM network. For example, in depletion load nMOS technology, any depletion mode transistor with VDD as the drain node can be assumed to be a pullup load, while all other transistors are generally strong transistors. If the extraction program also calculates the lengths and widths of the transistors, it can verify that the ratioed circuit will operate correctly by determining whether the highest resistance pulldown path can override the lowest resistance pullup path for nodes which have independent pullup and pulldown paths. If these worst case conditions are not satisfied, a warning message can be issued. By performing these static checks of the circuits, we avoid the need to check the pullup and pulldown ratios dynamically as the simulation proceeds.

As we generalize the switch-level model to include multiple levels of ratioing, charge sharing, and arbitrary use of weak transistors, the layout extraction becomes more difficult. The extraction program can calculate transistor resistances and node capacitances without great difficulty, but no general rule can take these parameters and assign transistor strengths and node sizes. For example, recognizing that the forceable inverter shown in Figure 2.2 requires three different transistor strengths would require a much more sophisticated algorithm than our rule for finding the MOSSIM network. Similarly, identifying which nodes will be sources of signals during charge sharing and which will be recipients cannot always be done with complete accuracy.

If we tailor the layout extraction program toward a particular class of designs and not try to utilize the full generality allowed by the model presented here, the extraction can be made reasonably straightforward and reliable. For example, circuits using more than two transistor strengths are relatively rare and even then appear only in limited configurations. The extraction program can look just for these configurations and for other portions of the design apply simple rules such as those used in deriving the MOSSIM network. Similarly, a bus node can usually be identified by its large capacitance relative to the

3. Logical Conductance Networks

3.1 Introduction

As was discussed in Chapter 2, the target state function provides a basic characterization of the dynamic behavior of a switch-level network. The value of this function was defined in terms of the set of possible steady state voltages for each node in a linear electrical network in the limit as the ratio parameter ρ approaches infinity. This definition helps clarify the relation between the switch-level model and MOS logic circuits but provides little aid in devising efficient simulation algorithms. In this chapter we start a transition from a circuit-oriented view of the switch-level model to a more abstract and logical view. A new form of network is introduced called *logical conductance* networks to focus attention on a key aspect of computing the target state. A logical conductance network represents a switch-level network with each transistor either nonconducting or fully conducting. The node states have values 0, 1, and X, but unlike switch-level networks, the network elements are "logical" conductances which can take on values from a small discrete set. The target state of a switch-level network can be defined in terms of the *steady states* of a set of logical conductance networks. The steady state of a logical conductance network is in turn defined in terms of the limiting case steady state voltages in an order of magnitude network with a unique set of parameters and initial conditions. Logical conductance networks provide an intermediate level of abstraction between electrical networks and switch-level networks.

3.2 Properties of the Electrical Model

We have defined the target state in terms of a class of linear time-invariant electrical networks containing voltage sources, capacitors with one side connected to ground, and linear resistors. The settings of the voltage sources are given by the vector **x**, the capacitances are given by the vector **c**, and the conductances of the resistors are given by the matrices **G** and **E**. The initial conditions of the network are defined by the vector **y**, giving the initial voltages on the capacitors. The nodes in this network

correspond to both the normal nodes and input nodes in the switch-level network. Some properties of these networks will be given here. Formal derivations of these properties are given in books on electrical network theory such as Desoer and Kuh [15]. They rely only on Kirchoffs' Current and Voltage Laws and on Ohm's Law.

In a linear network containing only passive, linear, time-invariant resistors and capacitors, any node connected by some conducting path to a voltage source (which has its negative terminal connected to GND) will have a steady state voltage uniquely determined by the voltage source settings and the resistor conductances. Such nodes are said to be *driven*. When node n_i is driven, its steady state voltage is a linear function of the voltage source settings $\mathbf{x}$:

$$v_i = \sum_{j=1}^{m} a_{ij} x_j.$$

The coefficients a_{ij} depend only on the resistor conductances and obey the following properties:

$$a_{ij} \geq 0.0, \text{ for all } j$$

$$\sum_{j=1}^{m} a_{ij} = 1.0.$$

These properties follow from the fact that for any possible set of voltage source settings given by the vector $\mathbf{x}$, v_i is bounded below and above by the minimum and maximum elements of $\mathbf{x}$, respectively. That is, if a_{ij} were negative for some value of j, then an out-of-range voltage would be obtained by setting x_j to a positive value and all other voltage sources to 0.0. Similarly, if the coefficients did not sum to 1.0, then an out-of-range voltage would be obtained by setting all voltage sources to the same nonzero value.

Nodes which have no conducting paths to voltage sources are said to be *charged*. Since every node has a nonzero capacitance and the network contains no floating capacitors, the steady state voltage of a charged node will be uniquely determined by the node capacitances, the resistor conductances (only whether each conductance is zero or nonzero), and the initial node voltages. When n_i is charged, its steady state voltage is a linear function of the initial node voltages $\mathbf{y}$:

$$v_i = \sum_{j=1}^{n} b_{ij} y_j.$$

The coefficients b_{ij} depend only on the resistor conductance and node capacitances and obey the following properties:

$$b_{ij} \geq 0.0, \text{ for all } j$$
$$\sum_{j=1}^{n} b_{ij} = 1.0.$$

These properties follow from the fact that for any possible set of initial node voltages $\mathbf{y}$, the steady state voltage on any charged node is bounded below and above by the minimum and maximum elements of $\mathbf{y}$, respectively.

A node is either charged or driven to its steady state voltage, and therefore if we adopt the convention that $a_{ij} = 0.0$ for all j when n_i is charged and $b_{ij} = 0.0$ for all j when n_i is driven, the two modes can be described by a single equation:

$$v_i = \sum_{j=1}^{m} a_{ij} x_j + \sum_{j=1}^{n} b_{ij} y_j. \tag{3.1}$$

The coefficients obey the following properties:

$$a_{ij} \geq 0.0, \text{ for all i and j}$$
$$b_{ij} \geq 0.0, \text{ for all i and j}$$
$$\sum_{j=1}^{m} a_{ij} + \sum_{j=1}^{n} b_{ij} = 1.0, \text{ for all i}$$
$$\sum_{j=1}^{m} a_{ij} \cdot \sum_{j=1}^{n} b_{ij} = 0.0, \text{ for all i.}$$

We will also be interested in networks where the elements of $\mathbf{E}$, $\mathbf{G}$, and $\mathbf{c}$ are given by continuous functions of the parameter ρ. In this case the coefficients a_{ij} and b_{ij} will be given by continuous functions of ρ. For any positive value of ρ, the network described by the conductance matrices $\mathbf{G}(\rho)$ and $\mathbf{E}(\rho)$ and the capacitance vector $\mathbf{c}(\rho)$ will be a passive, linear network with no floating capacitors, and

therefore the properties of the coefficients listed above must hold for all $\rho > 0.0$. Therefore, if we define a_{ij}^{∞} and b_{ij}^{∞} as the values

$$a_{ij}^{\infty} = \lim_{\rho \to \infty} a_{ij}(\rho)$$
$$b_{ij}^{\infty} = \lim_{\rho \to \infty} b_{ij}(\rho),$$

then

$$\lim_{\rho \to \infty} v_i = \sum_{j=1}^{m} a_{ij}^{\infty} \cdot x_j + \sum_{j=1}^{n} b_{ij}^{\infty} \cdot y_j$$

and the coefficients a_{ij}^{∞} and b_{ij}^{∞} obey the following properties:

$$a_{ij}^{\infty} \geq 0.0, \text{ for all i and j}$$
$$b_{ij}^{\infty} \geq 0.0, \text{ for all i and j}$$
$$\sum_{j=1}^{m} a_{ij}^{\infty} + \sum_{j=1}^{n} b_{ij}^{\infty} = 1.0, \text{ for all i}$$
$$\sum_{j=1}^{m} a_{ij}^{\infty} \cdot \sum_{j=1}^{n} b_{ij}^{\infty} = 0.0, \text{ for all i.}$$

These properties show that the limits used in the definition of the target state are mathematically well-defined.

3.3 Simplification of the Target State Definition

When a switch-level network contains transistors or nodes in the X state, the target state of a node is defined in terms of the set of possible steady state voltages for the node in an electrical network as the network parameters and initial conditions are varied over uncountably infinite sets. Fortunately, we need not evaluate all of these possibilities, because we only wish to know whether each set of voltages converges either to the set $\{ 0.0 \}$ or the set $\{ V_{dd} \}$ as ρ approaches infinity. If it can be determined that such a convergence does not occur, the target state is X. Thus we need not find the entire set of steady state voltages for all possible network parameters and initial conditions for each node, but only certain

properties about this set.

3.3.1 Node Voltages

First, let us consider the effects of having the voltage sources range over the settings $x^{min} \le x \le x^{max}$ and the initial node voltages range over $y^{min} \le y \le y^{max}$. Let the vectors x' and y' be any vectors satisfying the following requirements:

$$x^{min} \le x' \le x^{max} \text{ and } x^{min}_i < x^{max}_i \Rightarrow x^{min}_i < x'_i < x^{max}_i \tag{3.2}$$

$$y^{min} \le y' \le y^{max} \text{ and } y^{min}_j < y^{max}_j \Rightarrow y^{min}_j < y'_j < y^{max}_j \tag{3.3}$$

The following theorem shows that we need only evaluate the network for this single set of voltages.

Theorem 3.1:
For any conductance matrices G and E and capacitance vector c, if

$$V_f = \{ v(G, E, c, x, y) \mid x^{min} \le x \le x^{max},\ y^{min} \le y \le y^{max} \}$$

then $V_f = \{ V_{dd} \}$ if and only if $v(G, E, c, x', y') = V_{dd}$
$V_f = \{ 0.0 \}$ if and only if $v(G, E, c, x', y') = 0.0$

for any x' and y' satisfying equations 3.2 and 3.3.

Proof of Theorem 3.1:

We have already seen that the voltage v_i is given by a linear function of the elements of x and y as shown in equation 3.1, including in the limit as p approaches infinity. Furthermore, all coefficients are nonnegative and sum to 1.0. Clearly, v_i equals V_{dd} if and only if $y_j = V_{dd}$ for all j such that $a_{ij} > 0.0$, and $y_j = V_{dd}$ for all j such that $b_{ij} > 0.0$. Therefore, since x'_j equals V_{dd} if and only if $x^{min}_j = x^{max}_j = V_{dd}$ and similarly for y'_j, $v_i(G, E, c, x', y')$ equals V_{dd} if and only if $v_i(G, E, c, x, y)$

equals V_{dd} for all x such that $x^{min} \leq x \leq x^{max}$ and y such that $y^{min} \leq y \leq y^{max}$. The proof for $V_i = \{ 0.0 \}$ follows identically.∎

This theorem shows that even though the voltage formed by an input or normal node in the X state can range over an entire set of values $\{ v \mid 0.0 \leq v \leq V_{dd} \}$, we need only evaluate networks with this node having a unique voltage v such that $0.0 < v < V_{dd}$. This single value can be used to test the sensitivity of nodes in the network to this X state, where sensitivity is indicated by the values of the coefficients a_{ij} and b_{ij}. Thus we can simplify the definition of the target state from one in terms of entire sets of voltage parameters and initial conditions to one in terms of a single set x' and y'. Furthermore, the exact values of the voltages are unimportant, only whether they equal 0.0, V_{dd}, or lie between (exclusively) 0.0 and V_{dd}.

3.3.2 Conductance Matrices

When the switch-level network contains transistors in the X state, the target state is defined in terms of the steady state node voltages as the conductance matrices range over infinite sets, $G^{min} \leq G \leq G^{max}$ and $E^{min} \leq E \leq E^{max}$. Furthermore, while the elements of G^{min}, G^{max}, E^{min}, and E^{max} are functions of ρ, this property does not hold for all matrices in the set, which makes it difficult to establish the limiting case voltages. Observe, however, that rather than finding the entire set $V_i(\rho)$ for each node, if we could determine the minimum and maximum elements of the set, we would have sufficient information to find the target state. The following lemma demonstrates an important property of electrical networks as the resistors vary between their minimum and maximum values.

Lemma 3.1.
Suppose a passive, linear, time-invariant network contains a single variable resistor with conductance h. If $v_i(h)$ denotes the steady state voltage on node n_i as a function of h, then for any h, h^{min}, and h^{max} such that $0.0 \leq h^{min} \leq h \leq h^{max}$, either

$$v_i(h^{min}) \leq v_i(h) \leq v_i(h^{max})$$

or

$$v_i(h^{max}) \leq v_i(h) \leq v_i(h^{min}).$$

Proof of Lemma 3.1:

The proof must consider two cases.

First, suppose that node n_i is *charged* when $h = 0.0$, i.e. there is no conducting path from n_i to a voltage source. Then for nonzero h, n_i could be connected to more charged nodes than when $h = 0.0$, or a conductance path could exist from a voltage source to n_i and hence n_i is driven. Thus, there can be a discontinuity in v_i as a function of h at $h = 0.0$. For nonzero values of h, the steady state current through this resistor must equal 0.0, because this resistor is not contained in any loop in the network. As a result, the value of h can have no effect on the steady state voltages, and v_i must remain constant for any $h > 0.0$. Therefore, either $h = h^{min} = 0.0$ and $v_i(h^{min}) = v_i(h)$, or $h > 0.0$ and $v_i(h) = v_i(h^{max})$.

The more difficult case occurs when node n_i is *driven* for all values of h, i.e. there is always a conducting path to a voltage source. In Appendix I an equation is derived for the node voltage v_i as a function of h by an analysis of multi-port networks. This equation has the form

$$v_i(h) = v_i(0.0) + \frac{a\,h}{1.0 + b\,h} \tag{3.4}$$

where b is nonnegative. This equation indicates that v_i is approximately linear with respect to h for small h but approaches a constant asymptote as h becomes very large. Taking the derivative with respect to h gives

$$\frac{dv_i}{dh} = \frac{a}{(1.0 + b\,h)^2},$$

which equals 0.0 only in the trivial case where $a = 0.0$. Therefore v_1 has no minimum or maximum between $h = h^{min}$ and $h = h^{max}$ unless it is a constant function, and one of the two inequalities must hold.

∎

With this lemma we can show that the minimum and maximum steady state voltages for each node can be determined by considering only the minimum and maximum conductance values for each transistor.

Theorem 3.2.
Let E and G be the following sets of conductance matrices:

$$E = \{ E \mid e_{jk} = e^{min}_{jk} \text{ or } e^{max}_{jk} \}$$

$$G = \{ G \mid g_{jk} = g^{min}_{jk} \text{ or } g_{jk} = g^{max}_{jk} \text{ and } g_{jk} = g_{kj} \}.$$

If

$$V_1 = \{ v_j(G, E, c, x, y) \mid E^{min} \le E \le E^{max},\ G^{min} \le G \le G^{max},\ G = G^T \},$$

and

$$V_1' = \{ v_j(G, E, c, x, y) \mid E \in E,\ G \in G \},$$

then

$$minimum\ V_1 = minimum\ V_1' \tag{3.5}$$

$$maximum\ V_1 = maximum\ V_1'. \tag{3.6}$$

Proof of Theorem 3.2:

Suppose some pair of conductance matrices **G** and **E** give a minimum value for $v_i(\mathbf{G}, \mathbf{E}, c, x, y)$. For any element of **G** such that $g^{min}_{jk} < g_{jk} < g^{max}_{jk}$, Lemma 3.1 shows that we could set g_{jk} to one of the values g^{min}_{jk} or g^{max}_{jk} without affecting v_i, or else v_i would not have been a minimum value originally. This process can be repeated for all such g_{jk} until we have a matrix $G' \in G$ such that $v_i(\mathbf{G}, \mathbf{E}, c, x, y) = v_i(\mathbf{G}', \mathbf{E}, c, x, y)$. A similar process would find a matrix $E' \in E$ such that $v_i(\mathbf{G}, \mathbf{E}, c, x, y) = v_i(\mathbf{G}', \mathbf{E}', c, x, y)$, and therefore equation 3.5 must hold. A similar technique proves equation 3.6. ∎

This theorem shows that although transistors in the X state may create arbitrary conductances within some range, we can find the minimum and maximum node voltages by considering each transistor to be either nonconducting or fully conducting. In general, however, we must enumerate over all combinations of these two possibilities for all transistors in the X state, because the steady state voltage on a node can be minimized or maximized with some transistors nonconducting and others fully conducting. Furthermore, the voltages on different nodes can be minimized or maximized under different conditions. Thus, unlike the node voltages, we must still evaluate the network for a number of sets of conductance parameters, but this evaluation is now finite. Later we will show that the target state can be computed by a more direct method. Note also that this proof assumes that the variable conductance parameters are independent of one another, which stems from our original assumption that transistors with the same gate node will behave independently when that node is in the X state.

3.3.3 Revised Definition of the Target State

The results of Theorems 3.1 and 3.2 can be summarized by giving a new, but equivalent definition of the target state. Let $\mathbf{x}'$ and $\mathbf{y}'$ be any vectors satisfying equations 3.2 and 3.3 and define the sets $E(\rho)$ and $G(\rho)$ as

$$E(\rho) = \{ \mathbf{E} \mid e_{jk} = e^{min}_{jk}(\rho) \text{ or } e_{jk} = e^{max}_{jk}(\rho) \} \tag{3.7}$$

$$G(\rho) = \{ \mathbf{G} \mid g_{jk} = g^{min}_{jk}(\rho) \text{ or } g_{jk} = g^{max}_{jk}(\rho), \text{ and } g_{jk} = g_{kj} \}. \tag{3.8}$$

For any node n_i, let $V_i'(\rho)$ denote the set

$$V_i'(\rho) = \{ v_i(\mathbf{G}, \mathbf{E}, \mathbf{c}(\rho), \mathbf{x}', \mathbf{y}') \mid \mathbf{E} \in E(\rho), \ \mathbf{G} \in G(\rho) \}.$$

The matrices in the sets $E(\rho)$ and $G(\rho)$ have elements which are continuous functions of ρ. Therefore, we can take the limit of this set of possible steady state voltages as ρ approaches infinity to get the set

$$V_i'^{\infty} = \lim_{\rho \to \infty} V_i'(\rho).$$

The target state $\tilde{y}_i$ is then given as:

$$\tilde{y}_i = \begin{cases} 0, & V_i'^{\infty} = \{ 0.0 \} \\ 1, & V_i'^{\infty} = \{ V_{dd} \} \\ X, & \text{else.} \end{cases} \tag{3.9}$$

3.4 Rational Functions

In the formulation of the order of magnitude network model given in Chapter 2, a fully conducting transistor of strength γ_k is modeled by a linear resistor of conductance $\alpha\rho^k$. Similarly a normal node of size κ_k is modeled by a linear capacitor of capacitance $\alpha\rho^k$. In both cases, α is an arbitrary positive constant and can be different for different capacitors and resistors. Let us generalize this definition to model a fully conducting transistor of strength γ_k with a resistor of conductance $g(\rho)$ where g is an

arbitrary *rational function* of degree k such that $g(\rho) > 0.0$ for all $\rho > 0.0$. That is, g can be expressed by an equation of the form

$$g(\rho) = \frac{N(\rho)}{D(\rho)}$$

where N and D are both polynomial functions of ρ, and if N has degree n and D has degree d, then $k = n - d$. Similarly, we will model a normal node of size κ_k by a capacitor of capacitance $c(\rho)$ where c is an arbitrary rational function of degree k such that $c(\rho) > 0.0$ for all $\rho > 0.0$.

Observe that this generalization has no effect on our definition of the target state, because any rational function a of degree k can be expressed in the form

$$a(\rho) = \alpha\rho^k + b(\rho),$$

where α is a constant and $b(\rho)$ is a rational function of degree less than or equal to $k-1$. Therefore, for any $\varepsilon > 0.0$, there exists a constant ρ_0 such that

$$(\alpha-\varepsilon)\rho^k \leq a(\rho) \leq (\alpha+\varepsilon)\rho^k, \text{ for all } \rho > \rho_0.$$

A rational function of degree k behaves like the function $\alpha\rho^k$ as ρ becomes large. This generalization will assist our mathematical development, because the domain of rational functions has many of the properties of the domain of real numbers, i.e. they both form *fields* [27]. We can use expressions such as "$a+b$" to denote "the function of ρ which gives the value $a(\rho)+b(\rho)$" and assume that the + operation in this expression satisfies the usual properties of addition. We can also replace an interconnection of resistors by a single equivalent resistor in the order of magnitude model much as one can replace an interconnection of resistors having real-valued conductances by an equivalent resistor in ordinary electrical networks.

The degree of a rational function a can also be defined as the maximum value k such that

$$\lim_{\rho \to \infty} \frac{a(\rho)}{\rho^k} \neq 0.0$$

This function generalizes the usual idea of the degree of a polynomial function. Observe that $deg(0.0) = -\infty$. All other functions used here have integer-valued degrees. For any rational function a, we will use the notation a^{∞} to denote the value

$$a^{\infty} = \lim_{\rho \to \infty} a(\rho).$$

This notation will only be used when $deg(a) \leq 0$, and hence the limit is well-defined.

Let us define the domain $\mathfrak{F}$ as the set consisting of the constant function 0.0 along with all rational functions a such that $a(\rho) > 0.0$ for all $\rho > 0.0$. We will deal mainly with rational functions in this domain.

If $a, b \in \mathfrak{F}$ then

$$deg(a + b) = max(deg(a), deg(b)) \tag{3.10}$$

$$deg(a \cdot b) = deg(a) + deg(b) \tag{3.11}$$

$$deg(1.0/a) = -deg(a) \tag{3.12}$$

$$deg(a - b) \leq max(deg(a), deg(b)) \tag{3.13}$$

Note that when functions are subtracted, only a weak statement can be made about the degree of the resulting function, and furthermore $\mathfrak{F}$ is not closed under this operation.

As a final notation regarding rational functions, define the equivalence relation $\sim$ on rational functions as

$$a \sim b \text{ if and only if } deg(a) = deg(b).$$

The relations $\ll$ and $\gg$ are defined as

$$a \ll b \text{ if and only if } deg(a) < deg(b)$$

$$a \gg b \text{ if and only if } deg(a) > deg(b).$$

These relations are extended to vectors and matrices in the usual way, e.g. $\mathbf{a} \sim \mathbf{b}$ if and only if $a_i \sim b_i$ for all i. These relations characterize the limit of an expression which is often encountered in equations for node voltages in ratioed circuits. For any $a, b \in \mathfrak{F}$:

$$\lim_{\rho \to \infty} \frac{a(\rho)}{a(\rho)+b(\rho)} = \begin{cases} 1.0, & a \gg b \\ 0.0, & a \ll b \\ \alpha, & a \sim b, \end{cases} \tag{3.14}$$

where α is a constant such that $0.0 < \alpha < 1.0$.

3.5 Equivalent Networks

We are only interested in the steady state behavior of order of magnitude networks as ρ approaches infinity, and even then we need only a partial characterization of the node voltages. Thus many of the details of the electrical network can be ignored. This idea of ignoring certain details can be expressed in mathematical terms by defining equivalence relations such that networks which differ only in unimportant respects are equivalent.

The equivalence relation $\simeq$ is defined on elements of the set $\{ v \mid 0.0 \leq v \leq V_{dd} \}$ as $v \simeq v'$ if $v = v'$ or if $0.0 < v < V_{dd}$ and $0.0 < v' < V_{dd}$. This relation defines three equivalence classes: $\{ 0.0 \}$, $\{ V_{dd} \}$, and $\{ v \mid 0.0 < v < V_{dd} \}$, which correspond closely to the logic states 0, 1, and X. This relation is extended to vectors in the usual way, i.e. $\mathbf{v} \simeq \mathbf{v}'$ if $v_i \simeq v'_i$ for all i. The following lemma expresses the fact that the exact voltages on the nodes in the order of magnitude model are unimportant, only whether they equal 0.0, V_{dd}, or lie between (exclusively) 0.0 and V_{dd}.

Lemma 3.2.
For any conductance matrices **G** and **E** and capacitance vector **c**, if $\mathbf{x} \simeq \mathbf{x}'$ and $\mathbf{y} \simeq \mathbf{y}'$, then

$$\mathbf{v}(\mathbf{G}, \mathbf{E}, \mathbf{c}, \mathbf{x}, \mathbf{y}) \simeq \mathbf{v}(\mathbf{G}, \mathbf{E}, \mathbf{c}, \mathbf{x}', \mathbf{y}'). \tag{3.15}$$

The proof of this lemma closely follows the proof of Theorem 3.1. Let $\mathbf{v}$ denote the steady state node voltages for the vectors $\mathbf{x}$ and $\mathbf{y}$, and $\mathbf{v}'$ denote the steady state node voltages for the vectors $\mathbf{x}'$ and $\mathbf{y}'$. Referring to equation 3.1, v_i equals V_{dd} if and only if $x_j = V_{dd}$ for all j such that $a_{ij} > 0.0$ and $y_j = V_{dd}$ for all j such that $b_{ij} > 0.0$. Therefore, since x'_j (or y'_j) equals V_{dd} if and only if x_j (or y_j) equals V_{dd}, v'_i equals V_{dd} if and only if v_i equals V_{dd}. Similarly, v'_i equals 0.0 if and only if v_i equals 0.0, and by a process of elimination $0.0 < v'_i < V_{dd}$ if and only if $0.0 < v_i < V_{dd}$. ∎

We can also show that if the coefficients of the conductance and capacitance parameters in an order of magnitude network are varied, the steady state voltages will be equivalent under $\simeq$ as ρ approaches infinity.

Lemma 3.3.
Suppose an order of magnitude network contains a single variable resistor with conductance $\eta\rho^k$. If $v_i(\rho, \eta)$ denotes the steady state voltage on node n_i as a function of ρ and η, then for any constants $\eta_1, \eta_2 > 0.0$

$$\lim_{\rho \to \infty} v_i(\rho, \eta_1) \simeq \lim_{\rho \to \infty} v_i(\rho, \eta_2).$$

Proof of Lemma 3.3:

We have already seen that for a particular (positive) value of ρ, if n_i is *charged* when $\eta = 0.0$, its steady state voltage will be the same for any positive value of η, and therefore $v_i(\rho, \eta_1) = v_i(\rho, \eta_2)$ including in the limit as ρ approaches infinity. If n_i is *driven* for $\eta = 0.0$, then equation 3.4 can be applied to order of magnitude networks to give

$$v_i(\rho, \eta) = v_i(\rho, 0.0) + \frac{a(\rho) \cdot \eta\rho^k}{1.0 + b(\rho) \cdot \eta\rho^k}. \tag{3.16}$$

In the derivation of this equation in Appendix I it is also shown that $b(\rho) \geq 0.0$ for all ρ and that $deg(a) \leq deg(b) < 0$. Define $v_i^{\infty}(\eta)$ as

$$v_i^{\infty}(\eta) = \lim_{\rho \to \infty} v_i(\rho, \eta)$$

and consider the following three cases.

1. $deg(b) < -k$.

$$v_i^{\infty}(\eta) = \lim_{\rho \to \infty} v_i(\rho, 0.0),$$

which is independent of η.

2. $deg(b) > -k$.

$$v_i^{\infty}(\eta) = \lim_{\rho \to \infty} v_i(\rho, 0.0) + \lim_{\rho \to \infty} \frac{a(\rho)}{b(\rho)}$$

which is also independent of η.

3. $deg(b) = -k$.

$$v_i^{\infty}(\eta) = \lim_{\rho \to \infty} v_i(\rho, 0.0) + \frac{\alpha \eta}{1.0 + \beta \eta},$$

where

$$\alpha = \lim_{\rho \to \infty} a(\rho)\rho^k$$
$$\beta = \lim_{\rho \to \infty} b(\rho)\rho^k.$$

In this case the voltage depends on the value of η. The derivative of this function with respect to η is given by:

$$\frac{dv_i^{\infty}(\eta)}{d\eta} = \frac{\alpha}{(1.0 + \beta \eta)^2}$$

which equals 0.0 only if $\alpha = 0.0$, in which case $v_i^{\infty}(\eta)$ is a constant function. We know that

$0.0 \leq v_i^{\infty}(\eta) \leq V_{dd}$ for all values of η, and therefore if $v_i^{\infty}(\eta_1) = 0.0$ (or V_{dd}) for some value η_1, then

$$\left.\frac{dv_i^{\infty}(\eta)}{d\eta}\right|_{\eta=\eta_1} = 0.0,$$

in which case $v_i^{\infty}(\eta)$ is a constant function.

Combining these three cases, we can see that $v_i^{\infty}(\eta)$ must either be a constant function or else $0.0 < v_i^{\infty}(\eta) < V_{dd}$ for all $\eta > 0.0$. Therefore, the possible values of $v_i^{\infty}(\eta)$ for positive η must be equivalent under $\simeq$.∎

Lemma 3.4.
Suppose an order of magnitude network contains a single variable capacitor with capacitance $\eta\rho^k$. If $v_i(\rho, \eta)$ denotes the steady state voltage on node n_i as a function of ρ and η, then for any constants $\eta_1, \eta_2 > 0.0$

$$\lim_{\rho \to \infty} v_i(\rho, \eta_1) \simeq \lim_{\rho \to \infty} v_i(\rho, \eta_2).$$

Proof of Lemma 3.4:

Suppose the variable capacitor is associated with node n_j. This parameter will affect only the steady state voltages of nodes connected by some conducting path to n_j and then only if there is no conducting path from n_j to a voltage source. For any such node n_i:

$$v_i(\rho, \eta) = \frac{c(\rho)v_i(\rho, 0.0) + \eta\rho^k y_{ij}}{c(\rho) + \eta\rho^k}, \tag{3.17}$$

where c equals the sum of all capacitances connected by some path to n_j when $\eta = 0.0$. By considering three cases: $deg(c) < k$, $deg(c) > k$, and $deg(c) = k$, we can show that $v_i^{\infty}(\eta)$ must either be a constant function or else $0.0 < v_i^{\infty}(\eta) < V_{dd}$ for all positive values of η, much as in the proof of Lemma 3.3. ∎

These two lemmas can be extended to the case in which the conductance of the variable resistor or the capacitance of the variable capacitor is given by an arbitrary rational function of degree k. To see this, suppose the terms $\eta\rho^k$ in equations 3.16 and 3.17 are replaced by terms $\eta\rho^k + d(\rho)$, where d is a rational function of degree less than or equal to k-1. Then as we take the limit as ρ approaches infinity, all instances of d will drop out, leaving the original equations.

These two lemmas can be combined into a single result by defining the equivalence relation $\equiv$ between two order of magnitude networks N and N' as $N \equiv N'$ if and only if

$$G \sim G'$$
$$E \sim E'$$
$$c \sim c'$$
$$x \simeq x'$$
$$y \simeq y',$$

where the unprimed symbols refer to the network N and the primed symbols refer to N'.

Theorem 3.3.
Suppose order of magnitude electrical networks N and N' have steady state node voltages $v(\rho)$ and $v'(\rho)$, respectively. Then if $N \equiv N'$,

$$\lim_{\rho \to \infty} v(\rho) \simeq \lim_{\rho \to \infty} v'(\rho) \qquad (3.18)$$

The proof of this theorem follows directly from Lemmas 3.2, 3.3, and 3.4 (when extended to arbitrary rational functions) and the transitivity of the three equivalence relations. This theorem shows that although network equivalence is defined in terms of the structure of the network it also implies a form of behavioral equivalence. It also justifies our earlier statements that the coefficients of the order of magnitude network elements can be arbitrary positive constants without affecting the value of the target state.

3.6 Logical Conductance

As an aid in studying switch-level networks in which all transistors are either nonconducting or fully conducting, let us define a new type of network called *logical conductance* networks. Like a switch-level network, a logical conductance network contains input nodes and normal nodes, where each normal node n_i has a size $cap_i \in K = \{ \kappa_1, \ldots, \kappa_q \}$. Instead of containing transistors, however, these new networks contain logical conductances, which may have values in the set $\{ 0 \} \cup \Gamma = \{ 0, \gamma_1, \ldots, \gamma_p \}$. The correspondence between a switch-level network with all transistors either nonconducting or fully conducting and a logical conductance network is quite simple: a nonconducting transistor forms a logical conductance 0 (i.e. an open circuit), while a fully conducting transistor forms a logical conductance equal to its strength. Examples of the logical conductance networks corresponding to the switch-level model of an nMOS inverter are shown in Figure 3.1. A switch-level network with no transistors in the X state has a unique corresponding logical conductance network. When a switch-level network contains transistors in the X state, however, we must consider the logical conductance networks formed for all possible combinations of these transistors being either nonconducting or fully conducting. By focusing our attention on logical conductance networks, we temporarily set aside issues concerning transistors in the X state and solve a simpler problem. Once methods for analyzing logical conductance networks have been

Fig. 3.1. Examples of Logical Conductance Networks

developed, we will be able to generalize these methods to handle arbitrary switch-level networks.

The steady state behavior of a logical conductance network is defined in terms of the steady state voltages in an order of magnitude electrical network with a single set of network parameters and initial conditions. That is, a logical conductance of strength γ_k is modeled by a resistor of conductance $g(\rho)$, where g is an arbitrary rational function of degree k in the set $\mathcal{F}$. A normal node of size κ_k is modeled by a capacitor of capacitance $c(\rho)$ where c obeys the same requirements as g. Input node i_j in state x_j is modeled by a voltage source with voltage x_j, where $x_j = 0.0$ if $x_j = 0$, $x_j = V_{dd}$ if $x_j = 1$, and x_j equals some arbitrary voltage such that $0.0 < x_j < V_{dd}$ if $x_j = X$. When normal node n_j has an initial state y_j, the corresponding capacitor in the order of magnitude network is initially charged to a voltage y_j defined similarly.

The *steady state* of a logical conductance network is defined as the set of logic states (denoted with the vector $\bar{y}$) which the normal nodes would eventually reach when started in an initial state y. That is, if the network is modeled by an order of magnitude network with parameters $G(\rho)$, $E(\rho)$, $c(\rho)$, and x and initial conditions y, and v_i^{∞} is defined as

$$v_i^{\infty} = \lim_{\rho \to \infty} v_i(G(\rho), E(\rho), c(\rho), x, y)$$

then

$$\bar{y}_i = \begin{cases} 0, & v_i^{\infty} = 0.0 \\ 1, & v_i^{\infty} = V_{dd} \\ X, & 0.0 < v_i^{\infty} < V_{dd} \end{cases} \qquad (3.19)$$

The target state of a switch-level network can now be defined in terms of logical conductance networks rather than order of magnitude networks. From the matrices G and E in the order of magnitude model of a logical conductance network we can define two matrices G and E describing the logical conductances connecting the nodes in the network. That is, if $deg(g_{ij}) = k$, then $g_{ij} = \gamma_k$, and if $g_{ij} = 0.0$ then $g_{ij} = 0$, and similarly for E. Let the sets $\{E\}$ and $\{G\}$ equal the sets of logical

conductance matrices corresponding to the sets of order of magnitude conductance matrices *E* and *G* given by equations 3.7 and 3.8. These sets describe the set of logical conductance networks for all possible combinations of nonconducting and fully conducting transistors. Let $\bar{y}(G, E)$ denote the steady state of the logical conductance network with logical conductance matrices G and E, with the remaining parameters cap and x, and initial conditions y assumed implicitly. The target state can then be defined in terms of these steady states as

$$\bar{y}_i = \begin{cases} 1, & \bar{y}_i(G, E) = 1 \text{ for all } G \in \{ G \} \text{ and } E \in \{ E \} \\ 0, & \bar{y}_i(G, E) = 0 \text{ for all } G \in \{ G \} \text{ and } E \in \{ E \} \\ X, & \text{else.} \end{cases} \tag{3.20}$$

That is, the target state of a node will equal 0 or 1 if and only if it has this unique steady state regardless of variations in the conductances created by transistors in the X state. Note that if the switch-level network contains no transistors in the X state the sets { G } and { E } each contain one element and the target state equals the steady state of this unique logical conductance network.

If a logical conductance network can be modeled by an order of magnitude network N then it can also be modeled by some other network N′ if and only if $N \equiv N'$. Thus, there is a 1-1 correspondence between logical conductance networks and equivalence classes of order of magnitude networks. Logical conductance networks can be viewed as abstractions of order of magnitude networks, where those aspects of the structure which have no important effects on the behavior are ignored, as was shown in Theorem 3.3. The logic state $\bar{y}$ provides all of the information we require about the steady state voltages for the corresponding class of order of magnitude networks.

3.7 Properties of Logical Conductance Networks

With the analogy between logical conductance networks and classes of order of magnitude networks, we can derive some simple rules for interconnections of logical conductances. First, if logical conductances a and b are combined in series, then

$$g_{series} = min(a, b),$$

where logical conductance values are ordered

$$0 < \gamma_1 < \gamma_2 < \ldots < \gamma_F$$

To show this, suppose logical conductances a and b are modeled by resistors of conductance a and b where a and b are rational functions in the domain $\mathfrak{F}$. Then the net conductance g is given by

$$g = \frac{a\,b}{a + b},$$

and applying equations 3.10, 3.11, and 3.12 gives

$$deg(g) = deg(a) + deg(b) - max(deg(a), deg(b)) = min(deg(a), deg(b)).$$

Similarly, if logical conductances a and b are combined in parallel, then equation 3.10 gives

$$deg(g) = deg(a+b) = max(deg(a), deg(b)),$$

and therefore

$$g_{parallel} = max(a, b).$$

For more complex interconnections, the net logical conductance between two nodes equals the maximum logical conductance of all paths connecting them, where the logical conductance of a path is defined as the minimum logical conductance in the path. To see this, suppose that for a network of arbitrary linear resistors the net conductance between two terminals equals g. Let *P* denote some set of resistors forming a path between the two terminals and *C* denote some set of resistors which if removed would eliminate all paths between the two terminals. If g_j equals the conductance of resistor j, then

$$\frac{1_0}{\sum_P g_j^{-1}} \leq g \leq \sum_C g_j.$$

That is g is bounded below by the series conductance along any path and is bounded above by the parallel conductance through any cut-set. These inequalities can be shown by simple applications of loop and cut-set analysis. If these resistor conductances are given by rational functions of p, then for sufficiently large p, all conductance values will be partially ordered by the degree of their rational functions. Therefore

$$\min_P deg(g_j) \leq deg(g) \leq \max_C deg(g_j).$$

This can be expressed in terms of the corresponding logical conductances as

$$\min_P g_j \leq g \leq \max_C g_j,$$

where g is the net logical conductance between the two terminals and g_j is the logical conductance corresponding to resistor j. These inequalities hold for any path P and any cut-set C. Suppose P' is the path of maximum logical conductance, and C' is constructed by repeatedly removing the minimum element in the maximum uncut path until all paths between the two terminals are eliminated. Then the minimum element of P' equals the maximum element of C':

$$\min_{P'} g_j = g = \max_{C'} g_j.$$

Therefore g equals both the minimum logical conductance in the maximum path and the maximum logical conductance in the minimum cut-set. This rule illustrates how the switch-level model describes the operation of an MOS network in terms of only the dominant effects acting on each node. It considers only the strongest conductance path between two nodes and characterizes this path by a strength value from a small, discrete set. Much as rules for combining networks of linear resistors may not apply when voltage sources are present, we must be careful in applying these rules when input nodes are present.

We can also think of the normal nodes in a logical conductance network as forming *logical capacitances* with properties much like logical conductances. Logical capacitances can take on values in the set $K = \{ \kappa_1, \ldots, \kappa_q \}$ which are ordered

$$0 < \kappa_1 < \kappa_2 < \ldots < \kappa_q.$$

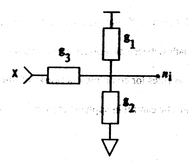
When logical capacitances a and b are connected in parallel through a nonzero conductance, they form an effective logical capacitance:

$$c_{parallel} = max(a, b).$$

Other forms of interconnection cannot occur, because the capacitors have one side tied to ground. This rule illustrates that the switch-level model considers only the largest capacitors connected to a node and characterizes the net capacitance by an element from a small, discrete set.

As a final set of rules, consider the logical conductance network shown in Figure 3.3, containing a normal node n_i connected by logical conductances g_1, g_2, and g_3 to input nodes in states 1, 0, and X, respectively. The steady state $\bar{y}_i$ will depend on the relative strengths of these conductances:

Fig. 3.2. State Formation in Ratioed Circuits



$$\bar{y}_i = \begin{cases} 1, & g_1 > max(g_2, g_3) \\ 0, & g_2 > max(g_1, g_3) \\ X, & else. \end{cases}$$

To show this rule, we can model the network by an order of magnitude network containing resistors of conductance $g_1(\rho)$, $g_2(\rho)$, and $g_3(\rho)$ connected to voltages V_{dd}, 0.0, and some voltage V such that $0 < V < V_{dd}$, respectively. Node n_i will have a steady state voltage

$$v_i(\rho) = \frac{g_1(\rho) \cdot V_{dd} + g_2(\rho) \cdot 0.0 + g_3(\rho) \cdot V}{g_1(\rho) + g_2(\rho) + g_3(\rho)}.$$

From equation 3.14 one can see that

$$v_i^{\infty} = \begin{cases} V_{dd}, & g_1 \gg g_2 + g_3 \\ 0.0, & g_2 \gg g_1 + g_3 \\ V', & else, \end{cases}$$

where $0.0 < V' < V_{dd}$. This rule illustrates that the logic state 0 (or 1) is formed on a driven node only when the connections to input nodes in state 0 (or 1) clearly dominate over all other connections to input nodes.

Similarly, if a set of normal nodes are connected such that the net logical capacitance of nodes initially in the 1 state equals c_1, of nodes initially in the 0 state equals c_2, and of nodes initially in the X state equals c_3, then the steady state of any node n_i in the set depends on the relative values of these logical capacitances:

$$\bar{y}_i = \begin{cases} 1, & c_1 > max(c_2, c_3) \\ 0, & c_2 > max(c_1, c_3) \\ X, & else. \end{cases}$$

This rule illustrates that the node state 0 (or 1) is formed on a charged node only when the net capacitance of nodes initially charged to 0 (or 1) clearly exceed the capacitance of all other nodes.

3.8 Summary

Logical conductance networks provide an abstraction of our electrical model of switch-level networks. The rather intractable definition of the target state in terms of electrical networks with parameters and initial conditions ranging over infinite sets can be reduced to one in terms of a finite set of logical conductance networks. The steady state of a logical conductance network is in turn defined in terms of an electrical network with a unique set of parameters of initial conditions.

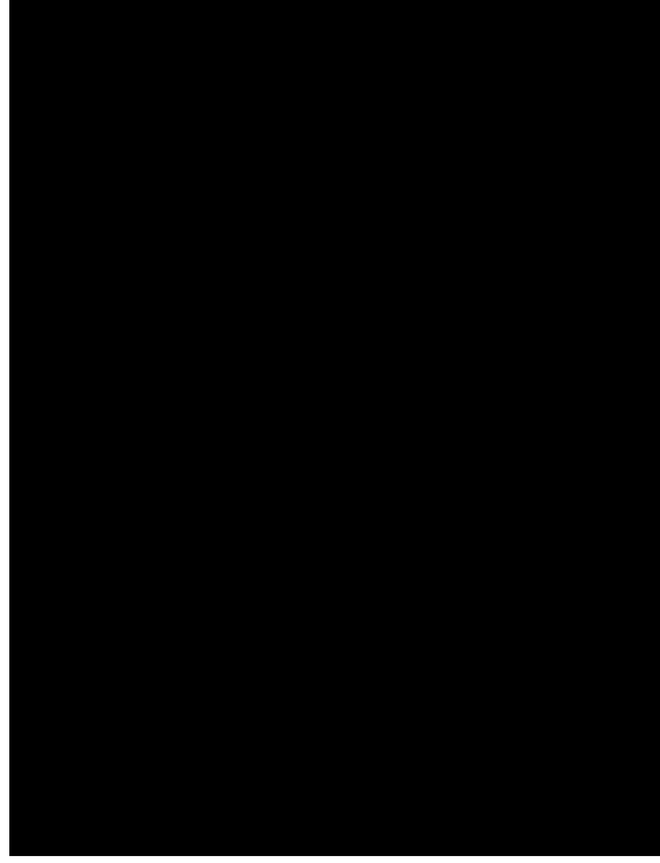
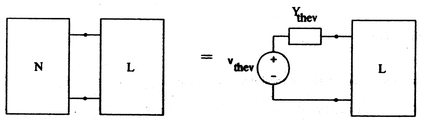


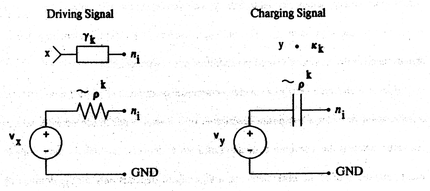
Fig. 4.1. Thevenin-Norton Equivalent Theorem



The Thevenin-Norton Equivalent Theorem [15], illustrated in Figure 4.1, states that for a linear network N connected to an arbitrary load L (i.e. some other network), we can replace N by its Thevenin (or Norton) equivalent network and obtain the same behavior. The Thevenin equivalent can be found for N without considering the nature of the load L. The voltage v_{thev} equals the voltage across the port when no load is attached and hence is called the *open-circuit* voltage. The admittance Y_{thev} equals the admittance across the port with no load attached and with all (independent) sources in N set to zero and hence is called the *zero-state* admittance. In our case setting all sources to zero involves short-circuiting the voltage sources corresponding to input nodes. Since we are concerned only with the steady state voltages we can consider the admittance to either be a conductance g_{thev} or a capacitance c_{thev}, because the conductance values will have an effect only when the node is driven to its steady state voltage, and the capacitance values will have an effect only when the node is charged. One can also see that a voltage source connected in series with a discharged capacitor is equivalent to the capacitor charged to that voltage.

Logic signals are related to Thevenin networks in two respects, as is shown in Figure 4.2. First, a logic signal describes the total behavior of a logical conductance at a particular node in terms of a single source of state, i.e. either an input node or a normal node, and a single network element, i.e. either a logical conductance or a logical capacitance. Second, a logic signal provides a direct model of the Thevenin equivalent of an order of magnitude network. That is, the logic signal at node n_i models the

Fig. 4.2. Relation Between Logic Signals and Thevenin Networks



Thevenin equivalent of the order of magnitude network as viewed at a port with positive terminal n_i and negative terminal GND. Thus we can prove properties about logic signals by demonstrating the analogous effect in the order of magnitude model.

The relation between a logic signal and the Thevenin equivalent of an order of magnitude network is defined as follows. For an order of magnitude network the open-circuit voltage v_{thev} and the zero-state conductance g_{thev} or capacitance c_{thev} are given by rational functions of ρ. If we let

$$v_{thev}^{\infty} = \lim_{\rho \to \infty} v_{thev}$$

then the logic signal corresponding to an order of magnitude network will have state 1 if $v_{thev}^{\infty} = V_{dd}$, state 0 if $v_{thev}^{\infty} = 0.0$, and state X if $0.0 < v_{thev}^{\infty} < V_{dd}$. The strength of the logic signal depends on the degree of g_{thev} or c_{thev}. A *driving* signal has strength in the set $\Gamma = \{\gamma_1, \ldots, \gamma_p\}$. A signal of strength γ_k corresponds to an order of magnitude network in which $deg(g_{thev}) = k$. This strength also equals the strength of the strongest path in the logical conductance network from the node to some input node. A *charging* signal has strength in the set $K = \{\kappa_1, \ldots, \kappa_q\}$. A signal of strength κ_k corresponds to an order of magnitude network in which $g_{thev} = 0.0$ and $deg(c_{thev}) = k$. This strength also equals the size of the largest node connected by some path in the logical conductance network. A *null* signal

represents an open circuit. That is, the corresponding order of magnitude network has zero-state conductance and capacitance equal to 0.0. As a consequence, the Thevenin voltage is indeterminate and is denoted by the logic state $\perp$. The null signal serves as an identity element when combining logic signals much as the number 0 is used in other domains of mathematics.

Just as a Thevenin network provides a composite description of a linear network at a particular port, a logic signal provides a composite description of a logical conductance network at a particular node. However, whereas the Thevenin equivalent of a linear network depends on the complete structure and exact parameters of all network elements, a logic signal depends only on the dominating effects at the node. The strength of a signal depends only on the strength of the maximum logical conductance path to an input node or on the size of the largest connected normal node, and the state depends only on the states of input nodes connected by maximum conductance paths or the states of the largest connected normal nodes. As a consequence, finding the logic signal for a logical conductance network will prove much easier than finding the Thevenin equivalent of a linear network.

4.3 Rules for Logic Signals

A simple set of rules describe the logic signals resulting when a logical conductance network is constructed by a series of primitive steps. These rules will first be listed and then shown to describe analogous effects in order of magnitude networks.

1. Formation

   a. Input node i_j forms a logic signal of strength γ_p and state x_j.

   b. Normal node n_j forms a logic signal of strength cap_j and state y_j.

2. Coupling

   A logic signal coupled through a nonzero logical conductance forms a signal with the state of the original signal and with strength equal to the minimum of the original signal strength and the conductance strength.

3. Combination

   Two logic signals can be combined into a single signal as follows:

   a. A stronger signal will override a weaker, and the weaker signal can be ignored completely.

   b. If the signals have the same strength and state, the resulting signal has this strength and state.

   c. If the signals have the same strength but different states, the resulting signal has this strength and state X.

4.3.1 The Formation Rule

The formation rule describes the logic signal formed by an isolated input or normal node. Comparing this rule to the Thevenin networks of Figure 4.2, one can see that the logic signal formed by input node i_j corresponds to a Thevenin network with v_{thev} set according to the logic state x_j, and with $deg(g_{thev}) = p$, where γ_p is the maximum allowable transistor strength. This resistor was not present in the formulation shown in Figure 2.4, but for the order of magnitude model it will behave just like an infinite conductance. That is, it acts as an identity element when resistors are combined in series and as an annihilator when resistors are combined in parallel. This avoids the need to add a special strength to represent an infinite conductance. The logic signal formed by normal node n_j corresponds to a Thevenin network with v_{thev} set according to the logic state y_j and if $cap_i = \kappa_k$ then $deg(c_{thev}) = k$. This

combination is equivalent to a capacitor with one side connected to GND and charged to the voltage v_{thev}.

4.3.2 The Coupling Rule

The coupling rule defines the effect of connecting a network described by a logic signal through a logical conductance to a node. Figure 4.3 shows how this rule describes the effect in the corresponding order of magnitude network. A driving signal of strength γ_k corresponds to a Thevenin network with the passive element having conductance g_1, where $deg(g_1) = k$. As was shown in the derivation of the series rule for logical conductances, when this element is connected in series with a conductance g_2, the net conductance has degree equal to $min(deg(g_1), deg(g_2))$. By the ordering of signal strengths, the connection rule describes this effect. A charging signal of strength κ_k corresponds to a Thevenin network with the passive element having capacitance c where $deg(c) = k$. This capacitor in series with a resistor of nonzero conductance will have a net conductance of 0.0 and a net capacitance of c. By our ordering of

Fig. 4.3. The Coupling Rule

signal strengths, a nonzero logical conductance strength will always exceed the strength of a charging signal, and hence the minimum of the signal strength and the logical conductance strength equals the signal strength.

4.3.3 Combination Rule

Fig. 4.4. The Combination Rule for Acyclic Connections

The combination rule describes the effect of connecting two networks described by logic signals to form a single network. First, let us assume the two networks are independent. Then this rule can be demonstrated by showing an analogous effect when ports of independent order of magnitude networks are connected. This involves three different cases as are shown in Figure 4.4, not counting the trivial cases where one of the signals is a null signal.

When two driving signals are combined, this corresponds to connecting a Thevenin network with voltage v_1 and conductance g_1 to one with voltage v_2 and conductance g_2. The zero-state conductance equals the effect of connecting the two resistors in parallel, and therefore

$$deg(g_{thev}) = max(deg(g_1, g_2)). \tag{4.2}$$

The combination rule yields a signal with strength equal to the maximum of the two signal strengths and hence correctly characterizes the value of g_{thev}. The open-circuit voltage is given by

$$v_{thev}(\rho) = \frac{g_1(\rho)v_1(\rho) + g_2(\rho)v_2(\rho)}{g_1(\rho) + g_1(\rho)}.$$

Therefore

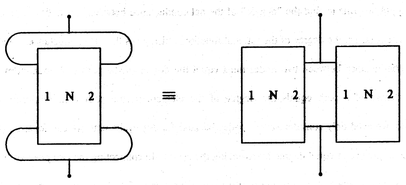
$$v_{thev}^{\infty} = \begin{cases} v_1^{\infty}, & g_1 \gg g_2 \\ v_2^{\infty}, & g_1 \ll g_2 \\ \alpha v_1^{\infty} + \beta v_2^{\infty}, & g_2 \sim g_2 \end{cases} \tag{4.3}$$

where α and β are positive constants. When $g_1 \sim g_2$, v_{thev}^{∞} will equal 0.0 (or V_{dd}) if and only if both v_1^{∞} and v_2^{∞} equal 0.0 (or V_{dd}). The combination rule yields a signal with state equal to the state of the stronger signal if the two signals have different strengths, and otherwise it yields state 0 (or 1) if and only if both signals have state 0 (or 1.) Thus the combination rule correctly characterizes the value of v_{thev}^{∞}. When two charging signals are combined, this corresponds to connecting a Thevenin network with voltage v_1 and capacitance c_1 to one with voltage v_2 and capacitance c_2. The analysis of this case proceeds much as with the previous case. When a driving signal is combined with a charging signal, this

corresponds to connecting a Thevenin network with voltage v_1 and conductance g_1 to one with voltage v_2 and capacitance c_2. The resulting network has a zero-state conductance g_1 and an open-circuit voltage v_1. Since a charging logic signal is always weaker than a driving signal, our rule correctly describes this effect.

We have shown that the combination rule holds when independent networks are combined. In fact a similar rule holds for arbitrary linear networks and hence we have not yet achieved a major simplification over more detailed electrical models. If the combination rule is applied only to independent networks, however, it can only be used to construct acyclic networks. In general, networks may contain cycles, and to construct these we must create cycles by combining the logic signals describing two nodes in the same network. This corresponds to connecting together two ports of the same order of magnitude network to form a single port. With logical conductance networks, the effect of this cyclic connection is given by simply applying the combination rule to the two signals. In other words, the port behavior of the order of magnitude network formed by connecting two ports of a network N is equivalent to the port behavior of the network formed by connecting the ports of two different copies of N, as is depicted in Figure 4.5. This can be motivated intuitively by noting that a logic signal describes only the dominating effects at a node and these effects involve only simple characterizations of acyclic paths. This

Fig. 4.5. The Combination Rule for Cyclic Connections



provides a major simplification over general linear network models, because the Thevenin-Norton Theorem applies only when the network N and the load L are independent. In general the Thevenin equivalent of a linear network can be found only by solving a system of linear equations.

To show the combination rule holds for cyclic connections, suppose ports 1 and 2 of an order of magnitude network N are connected. If one of these ports is described by a driving signal while the other is described by a charging signal, there can be no path in N between the positive terminals of the two ports. Therefore this case is just like an acyclic connection. Similarly, if both ports are described by charging signals, either there is no path in N between the positive terminals of the two ports, and hence it is just like an acyclic connection, or there is a path and the new connection is redundant. The combination rule correctly describes both of these possibilities. The more difficult case occurs when both ports are described by driving signals. We must show that the new network will have a zero-state conductance g_{thev} which obeys equation 4.2 and a limiting case open-circuit voltage v^{∞}_{thev} which obeys equation 4.3.

We can show the zero-state conductance obeys equation 4.2 using the rule derived in Chapter 3 for finding the net logical conductance between two nodes in a logical conductance network. This rule states that the net logical conductance equals the strength of the strongest path between the two nodes, where the strength of a path equals the weakest logical conductance in the path. Given the correspondence between the logical conductance γ_k and a resistor with conductance given by a rational function of degree k, we can apply this rule to find the "degree" of the net conductance between two nodes in an order of magnitude network, i.e. the degree of the rational function which gives the net conductance. The degree of the net conductance between two nodes must equal the degree of the path with maximum degree, where the degree of a path equals the degree of the element in the path with minimum degree. Furthermore, we need only consider *acyclic* paths, because for any cyclic path we can form a path with conductance of greater or equal degree by removing the cycles. In connecting the two ports of N, we do not form any new acyclic paths from the positive port terminals to GND, and hence the set of acyclic

paths across the new port equals the union of the sets of acyclic paths across ports 1 and 2 of N.

Therefore

$$deg(g_{tdev}) = max(deg(g_1), g_2)).$$

and the combination rule gives the correct signal strength for cyclic connections.

We can show that the limiting value of the open-circuit voltage v_{tdev}^{∞} obeys equation 4.3 by applying an equation for v_{tdev} derived in Appendix I by an analysis of multi-port networks:

$$v_{tdev}(\rho) = \frac{g_1(1.0-k_2)v_1 + g_2(1.0-k_1)v_2}{g_1(1.0-k_2) + g_2(1.0-k_1)} \qquad (4.4)$$

All of the terms in the above equation are rational functions of ρ. The factors k_1 and k_2 describe the strength of the connection within N between the positive terminals of the two ports. If both equal 0.0, then there is no connection and the equation reduces to the one for an acyclic connection. If $k_1(\rho)$ equals 1.0, then all paths from the positive terminal of port 2 to voltage sources pass through the positive terminal of port 1, and *vice-versa*. For values of $k_1(\rho)$ between (exclusively) 0.0 and 1.0, some paths from the positive terminal of port 2 to voltage sources pass through the positive terminal of port 1 and some do not. These factors obey the following properties for any positive value of ρ:

$$g_2(\rho)*k_1(\rho) = g_1(\rho)*k_2(\rho)$$
$$0.0 \leq k_1(\rho) \leq 1.0$$
$$0.0 \leq k_2(\rho) \leq 1.0$$
$$k_1(\rho) \cdot v_1(\rho) \leq v_2(\rho)$$
$$k_2(\rho) \cdot v_2(\rho) \leq v_1(\rho).$$

The second and third inequalities imply that k_1 and k_2 have degree less than or equal to 0. Let us consider 3 cases.

1. $g_1 \gg g_2$

Then $deg(k_2) = deg(g_2 * k_1 / g_1) < 0$, and hence $deg(1.0 - k_2) = 0$, while $deg(1.0 - k_1) \leq 0$. Therefore

$g_1(1.0-k_2) \gg g_2(1.0-k_1)$, and $v_{thev}^{\infty} = v_1^{\infty}$.

2. $g_1 \ll g_2$

An analysis similar to the previous case shows that $v_{thev}^{\infty} = v_2^{\infty}$.

3. $g_1 \sim g_2$

We wish to show that for some positive constants α and β

$$v_{thev}^{\infty} = \alpha v_1^{\infty} + \beta v_2^{\infty}.$$

Consider the following four possibilities:

a). $k_1^{\infty} < 1.0$, $k_2^{\infty} < 1.0$

Then $g_1(1.0-k_2) \sim g_2(1.0-k_1)$, and the desired result will clearly hold.

b). $k_1^{\infty} = k_2^{\infty} = 1.0$

This implies that $v_1^{\infty} = v_2^{\infty}$, in which case

$$v_{thev}^{\infty} = v_1^{\infty} = 0.5v_1^{\infty} + 0.5v_2^{\infty}.$$

c). $k_1^{\infty} < k_2^{\infty} = 1.0$

Then $v_{thev}^{\infty} = v_2^{\infty}$, and

$$k_1^{\infty} = \lim_{\rho \to \infty} (g_1(\rho)k_2(\rho)/g_2(\rho)) > 0$$

Furthermore $k_1^{\infty} \cdot v_1^{\infty} \leq v_{thev}^{\infty} \leq v_1^{\infty}$, and therefore

$$0.5k_1^{\infty} \cdot v_1^{\infty} + 0.5v_2^{\infty} \leq v_{thev}^{\infty} \leq 0.5v_1^{\infty} + 0.5v_2^{\infty}.$$

For $\beta = 0.5$ and for some α such that $0.0 < 0.5k_1^{\infty} \leq \alpha \leq 0.5$ the desired result must hold.

d). $k_2^{\infty} < k_1^{\infty} = 1.0$

The analysis of this case proceeds much like the previous one.

Therefore, when $g_1 \sim g_2$, v_{thev}^{∞} must depend on both v_1^{∞} and v_2^{∞}. This completes our proof that the combination rule correctly describes the effect of a cyclic connection.

4.4 The Steady State Signals

Each node n_i in a logical conductance network can be characterized by its *steady state signal*, denoted v_i, analogous to the Thevenin equivalent of the corresponding order of magnitude network at this node once it reaches steady state. The state of this signal equals $\bar{y}_i$, the steady state of the node. The value of this signal can be found by constructing the network by a series of primitive steps according to the three rules. This task can be difficult and tedious, however, and must be done separately for each node. Instead, we will show that an equation can be derived which expresses the value of the steady state signal for each node in terms of the signals formed by the input nodes and normal nodes in their initial states and by the steady state signals on other nodes.

Let us first introduce some notation, much of which will be replaced by more concise notation in Chapter 5 when the logic signal concept is formalized into an algebra of logic signals. For a nonzero strength value s, the expressions +s, -s, and xs denote signals with strength s and states 1, 0, and X, respectively. The null signal is denoted λ. Signal-valued variables are denoted with italic characters. The vector x denotes the signals formed by the input nodes in state x. That is x_i has state x_i and strength γ_{p_i}. Similarly, the vector y denotes the signals formed by the normal nodes in their initial state y. That is y_i has state y_i and strength cap_i. The signal resulting when logic signal a is coupled through logical conductance g is denoted $cple(a, g)$. According to the coupling rule this signal will have the same state as a and strength equal to the minimum of g and the strength of a. We will adopt a convention that $cple(a, 0) = \lambda$, i.e. any signal coupled through a zero conductance yields a null signal.

The rule for combining logic signals imposes the follow partial ordering on signal values:

That is, for signals a and b, $a \leq b$ if and only if the effect of combining a and b equals b. Thus, the result of combining a set of signals equals the *least upper bound* (abbreviated l.u.b.) of the set for this partial ordering. This partial ordering will prove important in our mathematical development. It provides a concise statement of the concept that the logic model considers only the dominating effects acting on a

node.

4.5 Constraints on the Steady State Signals

We can use the rules of logic signals to derive a set of constraints which the steady state signals must satisfy. Let N be an order of magnitude network in which node n_i has capacitance c_i, initial voltage y_i, and steady state voltage v_i. Suppose we were to connect an additional capacitor to this node with capacitance c where $c \sim c_i$, and charged to an initial voltage y, where $y \simeq y_i$, giving a network N'. Then the new network would be equivalent to the old one, i.e. $N \equiv N'$. Equivalent order of magnitude networks are represented by the same logical conductance network, and therefore if an analogous process is performed with a logical conductance network, the logic signals should remain unchanged. This new capacitor is described by a logic signal equal to y_i, and the addition of the capacitor is described by an application of the combination rule to y_j and η_j giving

$$\eta_i = \text{l.u.b.}\{y_i\, \eta_j\} \geq y_i.$$

Similarly, suppose that N has a total conductance e_{ij} connecting node n_i to the voltage source corresponding to input node i_j. Then if a conductance e is added between n_i and this voltage source such that $e \sim e_{ij}$, the new network N' will be equivalent, i.e. $N \equiv N'$. Therefore, if we perform an analogous process with a logical conductance network, the logic signals should remain unchanged. A resistor with order of magnitude conductance e is described by the logical conductance e_{ij}, and hence the effect of this new conductance on n_i is described by the signal $cpl\epsilon(x_j, e_{ij})$, giving

$$\eta_i = \text{l.u.b.}\{cpl\epsilon(x_j, e_{ij}),\, \eta_j\} \geq cpl\epsilon(x_j, e_{ij}).$$

This result holds even if $e_{ij} = 0$, because $cpl\epsilon(x_j, 0) = \lambda$, and $\eta_i \geq \lambda$. By a similar line of reasoning

$$\eta_i = \text{l.u.b.}\{cpl\epsilon(\eta_j, e_{ij}),\, \eta_j\} \geq cpl\epsilon(\eta_j, e_{ij}).$$

If v_i is greater than or equal to all of these signals, it must be greater than or equal to their least upper bound:

$$v_i \geq \text{l.u.b.}\Big(\{y_i\} \cup \{cple(x_j, c_{ij}) \mid 1 \leq j \leq m\} \cup \{cple(v_j, g_{ij}) \mid 1 \leq j \leq n\}\Big).$$

If we define the function f_i as

$$f_i(a) = \text{l.u.b.}\Big(\{y_i\} \cup \{cple(x_j, c_{ij}) \mid 1 \leq j \leq m\} \cup \{cple(a_j, g_{ij}) \mid 1 \leq j \leq n\}\Big),$$

then these constraints are expressed by an equation $v_i \geq f_i(v)$. If we then let f denote the function yielding a vector with ith element equal to the application of f_i to the argument, then the set of constraints for all node signals can be expressed by a single equation $v \geq f(v)$.

We can show in fact that $v = f(v)$. In other words, the steady state signal on each node equals the least upper bound of the signal formed initially on the node and the signals on adjacent input and normal nodes coupled through the logical conductances between them. To show this, let v' denote the vector given by $v' = f(v)$, and let r and r' denote the vectors of signal strengths for the signal vectors v and v'. That is, for any value of i, r_i equals the strength of v_i and r'_i equals the strength of v'_i. Suppose first that for some n_i, v_i is a charging signal, i.e. it has strength $r_i \in K$. Then for any n_j such that $g_{ij} > 0$,

$$v_i \geq v'_i \geq cple(v_j, g_{ij}) = v_j \geq cple(v_i, g_{ij}) = v_i,$$

which implies that $v_i = v'_i$. If, on the other hand, $g_{ij} = 0$ for all j, then node n_i is completely isolated and therefore $v_i = v'_i = y_i$. Next, suppose v_i is a driving signal, i.e. it has strength $r_i \in \Gamma$. If $v_i > v'_i$ then either $r_i > r'_i$ or v_i has state X and v'_i has state 0 or 1. If $v_i > v'_i$, then for any normal node n_j, the constraints on the logic signals imply the following constraints on the signal strengths:

$$r_i > r'_i \geq min(g_{ij}, r_j)$$
$$r_j \geq r'_j \geq min(g_{ij}, r_i)$$

Therefore, if $r_j \geq r_i$, the first inequality can hold only if $g_{ij} \leq r'_i < r_i$, and if $r_j < r_i$, the second inequality

can hold only if $\delta_{ij} \leq \gamma_j < \gamma_i$. Similarly, for any depot node j,

$$\gamma_i > \gamma_j \geq \max_k \delta_{jk} \gamma_k = \eta_j$$

In other words, all logical conductances connected to x_i have strength less than γ_i, the strength of the steady state signal. Therefore, since any path from x_i to an depot node must pass through one of these conductances it cannot have strength equal to the steady-state signal strength. This contradicts the fact that the strength of the steady state signal equals the maximum of those path strengths, and therefore γ_i cannot be greater than γ_i. Thus we can assume that $\gamma_i = \gamma_i$ for all i. Now suppose $\gamma_i = \gamma_i$ but the state of x_i equals X and the state of x_i equals 0. For any node x_k for $\delta_{ik} \geq \gamma_i$, then $\gamma_i \geq \min(\delta_{ik}, \gamma_k) = \gamma_i$. Furthermore, $\gamma_i \geq \max_k \delta_{ik} \gamma_k$ and therefore x_i must have state $\overline{\gamma}_i = X$. By a similar line of reasoning, any node x_j for which σ_{ij} equals γ_i (it cannot be greater) $\gamma_j = 0$. In other words, any node connected to x_i by a logical conductance of strength greater than or equal to γ_i must have state 0. Let us look at what this implies in the corresponding order of magnitude network. By Kirchoff's Current Law, the net current flowing out from node x_i equals 0.0:

$$\sum_{j=1}^{m} (\gamma_i \cdot \gamma_j) \sigma_{ij} + \sum_{j=1}^{n} (\gamma_i \cdot \gamma_j) \sigma_{ij} = 0.0.$$

In the above equation all terms are rational functions of ρ. Assuming $\gamma_i = \gamma_i$, we can divide through by ρ^k and take the limit as ρ approaches infinity:

$$\lim_{\rho \to \infty} \left(\sum_{j=1}^{m} (\gamma_i \cdot \gamma_j) \frac{\sigma_{ij}}{\rho^k} + \sum_{j=1}^{n} (\gamma_i \cdot \gamma_j) \frac{\sigma_{ij}}{\rho^k} \right) = 0.0.$$

Let us look at the individual terms in this equation. For any j such that $\deg(\sigma_{ij}) < k$,

$$\lim_{\rho \to \infty} (\gamma_i \cdot \gamma_j) \frac{\sigma_{ij}}{\rho^k} = (\gamma_i \cdot \gamma_j) \lim_{\rho \to \infty} \frac{\sigma_{ij}}{\rho^k} = 0.0.$$

For any j such that $\deg(\sigma_{ij}) \geq k$, $\overline{\gamma}_j = 0$, and therefore $\gamma_j^{(\infty)} = 0.0$. Furthermore, by our assumption $\overline{\gamma}_i = X$, and therefore $\gamma_i^{(\infty)} > 0.0$. This implies that

$$\lim_{\rho \to \infty} (\gamma_i \cdot \gamma_j) \frac{\sigma_{ij}}{\rho^k} = (\gamma_i \cdot \gamma_j^{(\infty)}) \lim_{\rho \to \infty} \frac{\sigma_{ij}}{\rho^k} > 0.0.$$

A similar line of reasoning shows that for any j if $deg(e_{ij}) < k$,

$$\lim_{\rho \to \infty} (v_i - x_j) \frac{e_{ij}}{\rho^k} = 0.0,$$

and if $deg(e_{ij}) \geq k$,

$$\lim_{\rho \to \infty} (v_i - x_j) \frac{e_{ij}}{\rho^k} > 0.0.$$

These results would imply that the above summation is greater than 0.0, which is not possible. Therefore v_i cannot have state X when v'_i has state 0, and a similar line of reasoning shows that v_i cannot have state X when v'_i has state 1. This completes our proof that $v = f(v)$.

We have shown that the rules for logic signals imply a set of constraints which must be satisfied by the steady state signal for each node n_i:

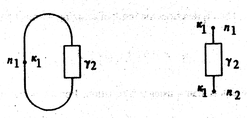
$$v_i = \text{l.u.b.}\left(\{ y_i \} \cup \{ cple(x_j, c_{ij}) \mid 1 \leq j \leq m \} \cup \{ cple(v_j, g_{ij}) \mid 1 \leq j \leq n \} \right). \qquad (4.5)$$

Note how we used the fact that the combination rule holds for cyclic as well as acyclic connections to derive the constraints on the steady state signal for each normal node in terms of the steady state signals on other normal nodes. No such set of constraints can be given for the Thevenin equivalents at the different ports of an arbitrary linear network, because cyclic combinations cannot be dealt with and because there is no analogous partial ordering. Thus, we have taken a major step away from electrical network concepts.

4.6 Specification of the Steady State Signals

The constraints on the steady state signals given in equation 4.5 can be expressed by a recurrence equation of the form $v = f(v)$. This equation in itself, however, does not provide a unique specification of the steady state signals, because it may have multiple solutions. For example, the networks in Figure 4.6 contain no input nodes, and assuming the nodes have initial state X, the steady state signals of the nodes in both cases should equal $x\varepsilon_1$. Suppose in the first example, however, that we let a equal $+\gamma_2$.

Fig. 4.6. Networks with Extraneous Solutions



Then, since g_{ii} equals γ_2, and $+\gamma_2 = cpl(+\gamma_2, \gamma_2)$, the signal a satisfies the recurrence relation $a = f(a)$, as would any signal such that $x\kappa_1 \leq a \leq x\gamma_2$. The second example shows a similar case in which two nodes have mutually dependent signals, and hence any solution such that $x\kappa_1 \leq a_1 = a_2 \leq x\gamma_2$ would satisfy the recurrence relation $a = f(a)$. These examples demonstrate that the equation $a = f(a)$ may have solutions in which some nodes have signals greater than the steady state signals because of extraneous cyclic dependencies allowed by the recurrence relation. In the second case, a single logical conductance creates cyclic dependencies in both directions. These extraneous solutions have no physical significance; they are artifacts of the simplified view of electrical networks provided by logic signals.

Fortunately, we can exclude these extraneous solutions by considering only the *minimum* vector which satisfies the recurrence relation. Let v' denote the minimum solution of the equation $a = f(a)$. That is, $v' = f(v')$ and for any a such that $a = f(a)$, $v'_i \leq a_i$ for all i. We will show in Chapter 6 that such a unique minimum solution must exist. The minimum solution depends only on the initial signals on the nodes and the consistency constraints imposed by the recurrence relation. Therefore, it seems quite reasonable that the vector of steady state signals v should equal the minimum solution v'. By a generalization of the technique used to show that $v = f(v)$, we will show that $v = v'$. This gives us a complete specification of the steady state signals in terms of a simple set of operations on logic signals.

First suppose for some node n_i, γ_i is a charging signal. Let A equal the set of all nodes connected by some conducting path to n_i. Then γ_i describes the result of charging starting between these nodes:

$$\gamma_i = \text{l.u.b.}\{\gamma_j \mid n_j \in A\}.$$

We can show that v'_i cannot be less than γ_i using this equation. For any n_j and n_k in A such that $g_{jk} \geq 0$,

$$v'_j \geq \mathit{cpfl}(g_{jk}, v'_k) = v'_k \geq \mathit{cpfl}(g_{jk}, v'_j) = v'_j,$$

and therefore $v'_j = v'_k$. For any node $n_j \in A$, we can show by induction on the length of the shortest path to n_j that $v'_i = v'_j$. Furthermore $v'_j \geq \gamma_j$ for all n_j which implies that

$$v'_i \geq \text{l.u.b.}\{\gamma_j \mid n \in A\} = \gamma_i.$$

Therefore $\gamma_i = v'_i$ for any charged node.

Next, let us consider driving signals. Let r and r' denote the vectors of signal strengths giving the strengths of the elements of r and r', respectively. For any node n_j, if $\gamma_i > v'_j$, either $r_i > r'_i$ or γ_i has state X and v'_i has state 0 or 1. Let $s \in \Gamma$ equal a strength value where for all J such that $r_j > s$, $r_j = r'_j$, but for some L $r_l = s > r'_l$. Let A and B denote the sets

$$A = \{n_j \mid s = r_j > r'_j\}$$
$$B = N - A.$$

Then for any $n_q \in A$ and $n_p \in B$:

$$s > r'_i \geq \mathit{min}(g_{qp}, r'_j)$$
$$r_j \geq \mathit{min}(g_{qp}, r_j).$$

Therefore, if $r_j > s$, then $r'_j = r_j$ and the first inequality can hold only if $g_{ij} < s$. Similarly, if $r_j = s$, and since $n_j \notin A$, then $r'_j = r_j$ and the first inequality can hold only if $g_{ij} < s$. Finally, if $r_j < s$, the second inequality can hold only if $g_{ij} < s$. Furthermore, for any input node i

$$s > r_i \geq mnk(c_{ij}, r_j) = c_{ij}$$

and therefore $c_{ij} < r_i$. In other words all logical conductances connecting nodes in A to input or normal nodes outside of A have strength less than s, the strength of the steady state signals for all nodes in A. Any path from a node in A to an input node must pass through one of these conductances and hence cannot have strength equal to the signal strength, but this contradicts the fact that the strength of the steady state signal equals the maximum of these path strengths. Therefore, there can be no strength value s satisfying our requirement and $r_i = r'_i$ for all i. Next, let s equal a strength value where for all j such that $r_j > s$, $v_j = \overline{v}_j$, but for some i, $r_i = s$ and $v_i > \overline{v}_i$. Let A and B denote the following sets:

$$A = \{ n_i \mid r_i = s, v_i \text{ has state } x, \text{ and } \overline{v}_i \text{ has state } 0 \}$$
$$B = N - A.$$

Then for any $n_i \in A$ and $n_j \in B$ if $g_{ij} \geq r_i$, then $r_j \geq mnk(g_{ij}, r_i) = r_i$. Furthermore, $v_i \geq cpkt(v_j, g_{ij})$, and therefore v_j must have state $\overline{v}_j = 0$. By a similar line of reasoning, for any $n_i \in A$ and any input node j for which c_{ij} equals r_i (it cannot be greater), $x_{ij} = 0$. This shows that for any normal node in B or for any input node, if it is connected to a node in A by a logical conductance of strength greater than or equal to s, it must have state 0. Let us look at what this implies in the corresponding order of magnitude network. By Kirchoff's Current Law, if we form a cut-set consisting of all branches connecting nodes in A to nodes in B and to voltage sources corresponding to input nodes, the net current through this cut-set must equal 0.0:

$$\sum_{n_i \in A} \sum_{j=1}^{m} (v_i - x_j)g_{ij} + \sum_{n_i \in A} \sum_{n_j \in B} (v_i - v_j)g_{ij} = 0.0.$$

In the above equation, all terms are rational functions of ρ. For $s = \gamma_k$, we can divide through by ρ^k and take the limit as ρ approaches infinity,

$$\lim_{\rho \to \infty} \left(\sum_{n_i \in A} \sum_{j=1}^{m} (v_i - x_j)\frac{g_{ij}}{\rho^k} + \sum_{n_i \in A} \sum_{n_j \in B} (v_i - v_j)\frac{g_{ij}}{\rho^k} \right) = 0.0.$$

Let us look at the individual terms in this equation. For any i and j such that $deg(g_{ij}) < k$,

$$\lim_{\rho \to \infty} (v_i - v_j)\frac{g_{ij}}{\rho^k} = (v_i^{\infty} - v_j^{\infty}) \lim_{\rho \to \infty} \frac{g_{ij}}{\rho^k} = 0.0.$$

For any j such that $n_j \in B$ and $deg(g_{ij}) \geq k$, $\bar{y}_j = 0$, and therefore $v_j^{\infty} = 0.0$. Furthermore, we have assumed that for any $n_i \in A$, $\bar{y}_i = x$, and therefore $v_i^{\infty} > 0.0$. This implies that

$$\lim_{\rho \to \infty} (v_i - v_j)\frac{g_{ij}}{\rho^k} = (v_i^{\infty} - v_j^{\infty}) \lim_{\rho \to \infty} \frac{g_{ij}}{\rho^k} > 0.0.$$

A similar line of reasoning shows that if $deg(e_{ij}) < k$,

$$\lim_{\rho \to \infty} (v_i - x_j)\frac{e_{ij}}{\rho^k} = 0.0,$$

and if $deg(e_{ij}) \geq k$,

$$\lim_{\rho \to \infty} (v_i - x_j)\frac{e_{ij}}{\rho^k} > 0.0.$$

These results would imply that the above summation is greater than 0.0, which is not possible. Therefore the set A must be empty, but a similar result holds if A is defined as

$$A = \{ n_i \mid r_i = s,\ v_i \text{ has state } X,\ \text{and } v'_i \text{ has state } 1 \},$$

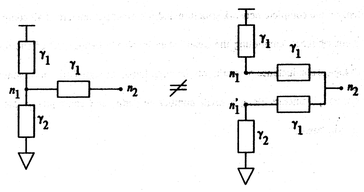
and hence there can be no such strength value s. For all driven nodes $v_i = v'_i$, which completes our proof that the vector of steady state signals v will equal the unique minimum value satisfying the recurrence relation $v = f(v)$.

The constraints on the steady state signals given by equation 4.5 give us a specification of the set of steady state signals without any reference to an electrical model, as long as we consider only the minimum solution. This will allow us to develop a method of computing the steady state of a logical conductance network without evaluating any electrical networks.

4.7 Signal Blocking

Informally, we can view a driving logic signal as describing the combined effect of those input nodes connected by a path of maximum strength. Under certain conditions, however, this informal view may not be entirely accurate, because of a phenomenon we call *signal blocking*. Consider, for example, node n_2 in the two networks shown in Figure 4.7. In both networks there is a path of strength $min(\gamma_1, \gamma_1) = \gamma_1$ to an input node in state 1, and a path of strength $min(\gamma_2, \gamma_1) = \gamma_1$ to an input node in state 0. Our informal view would then suggest that in both networks n_2 = l.u.b.$\{+\gamma_1, -\gamma_1\} = X\gamma_1$, and hence node n_2 has a steady state X. While this analysis yields the correct result for the second network, it fails for the first. In the first network node n_1 will be driven to 0 by the signal $-\gamma_2$ and hence n_2 will also be driven to 0 by the signal $cple(-\gamma_2, \gamma_1) = -\gamma_2$. This example demonstrates that when the paths to input nodes are not independent, some signals may be blocked along a path by a signal of greater strength. For example the signal $+\gamma_1$ is blocked at n_1 by the signal $-\gamma_2$, and hence node n_2 is unaffected by this signal. Our informal view does not take such a possibility into account, although our more formal method does. Note that this phenomenon can be ignored in *restricted* logical conductance networks, which are defined as networks in which any logical conductance between two normal nodes must have strength γ_p. This class of networks can be used for analyzing restricted switch-level networks as were

Fig. 4.7. Signal Blocking Example



defined in Chapter 2. In restricted networks, the strength of any path to an input node will be determined by the strength of the final logical conductance in the path, and hence the informal rule will correctly describe the precedence between signal paths.

Signal blocking creates difficulties in the formulation of a method for computing the steady state signals, but a resolution of this problem leads to an efficient method for computing the target state of a switch-level network.

4.8 Conclusion

The simple set of rules for logic signals lead to a specification of the steady state signals:

$$r_i = \text{l.u.b.}\left(\{r_i\} \cup \{ cpic(x_j, e_{ij}) \mid 1 \le j \le m \} \cup \{ cpic(r_j, g_{ij}) \mid 1 \le j \le n \}\right). \qquad (4.5)$$

where the elements of r must be the minimum set of signals satisfying the above equation for all i. Since the steady state $\overline{y}_i$ equals the state of the signal r_i, this gives us a specification of the steady state for any logical conductance network in terms of a simple set of operations on logic signals rather than in terms of an electrical model. Thus we have completed our transition form a circuit-oriented view of the switch-level model to a more abstract logical view. From this point onward we will deal only with these logical concepts. The order of magnitude electrical network model has served out its useful life.

The logic signal formalism takes advantage of our simplified model of the electrical behavior of switch-level networks. In general linear networks, the state variables (i.e. node voltages) are to some degree dependent on the complete network structure and the exact parameters of all connected network elements. As a consequence, computing the steady state involves solving a set of simultaneous linear equations. With logical conductance networks, on the other hand, the signal on a node generally depends only on the network parameters along a small number of paths, and consequently the state can be computed much more easily.

Many logic simulation programs use values (generally called "states") which describe both logic state and relative precedence. Such values corresponded closely to logic signals. For example the high impedance or H state described in Chapter 1 corresponds closely to the null signal λ, because both represent open circuits and are overridden by any other state. More MOS logic simulators [9, 34, 5] encode both logic state and strength into a single value and simulate the network by forming, combining, and propagating these values according to some set of rules. These rules, however, often do not display the consistency, accuracy, and generality required for a more formal treatment. The rules for logic signals outlined in this chapter, on the other hand, can be formalized into an algebra with operations corresponding to signal combination and coupling, and algorithms can be developed which perform simulations by solving equations in this algebra.

5. An Algebra of Logic Signals

5.1 Introduction

Shannon showed originally that Boolean algebra could be applied to the study of relay networks, and later this algebra was applied to logic gate networks. With MOS networks, however, Boolean algebra does not suffice. Although the nodes assume Boolean (or ternary) logic states, the network conditions which create these states depend on the relative sizes of conductances and capacitances. Therefore, the conductances and capacitances cannot simply be characterized by Boolean values. Furthermore, in a voltage-driven logic, one must consider the state of a signal source as well as the strength of the conductance path to it. We will develop our own algebra based on the rules of logic signals which retains much of the simplicity of Boolean algebra while allowing a more detailed description of the network.

5.2 General Definitions

Let us start by defining some terminology, most of which is consistent with standard mathematical practice.

For domains $\mathfrak{D}$ and $\mathfrak{D}'$ and a function $f:\mathfrak{D} \rightarrow \mathfrak{D}'$ define the *pointwise extension* to vectors of size n, i.e. $f:\mathfrak{D}^n \rightarrow \mathfrak{D}'^n$ as the vector resulting from the application of the function to each component of the argument. The pointwise extension of a function with more than one argument is defined as the vector resulting from the application of the function to the corresponding components of each argument. The pointwise extension to matrices of size $n \times m$ is defined similarly. For example, in linear algebra matrix addition is the pointwise extension of scalar addition. Whenever a scalar function is shown applied to vector or matrix arguments, its pointwise extension is implied.

For a domain $\mathfrak{D}$ a partial ordering $\leq$ is extended to vectors in $\mathfrak{D}^n$ and matrices in $\mathfrak{D}^{n\times m}$ by saying that one vector (or matrix) is $\leq$ another if the ordering holds for each pair of corresponding elements. Observe that the pointwise extension of the least upper bound operation for some partial ordering equals the least upper bound operation for the extension of the partial ordering.

For a domain $\mathfrak{D}$ with partial ordering $\leq$ and a domain $\mathfrak{D}'$ with partial ordering $\leq'$, a function $f:\mathfrak{D} \rightarrow \mathfrak{D}'$ is *monotonic* if

$$a \leq b \Rightarrow f(a) \leq' f(b)$$

A function of more than one argument is monotonic if it is monotonic for each argument. One can easily see that the composition of monotonic functions must be monotonic, as is the pointwise extension of a monotonic function.

5.3 The Algebra of Signal Strengths

Logic signals have strengths in the finite set

$$\mathfrak{D} = \{0, \kappa_1, \ldots, \kappa_q, \gamma_1, \ldots, \gamma_p\}$$

which are totally ordered:

$$0 < \kappa_1 < \ldots < \kappa_q < \gamma_1 < \ldots < \gamma_p$$

When two signals combine, the resulting signal has strength equal to the maximum of the two signal strengths. When a signal is coupled through a logical conductance, the resulting signal has strength equal to the minimum of the signal and conductance strengths. The binary operations $\uparrow$ and $\downarrow$ are defined to give the maximum and minimum of their arguments, i.e.

$$a \uparrow b = max(a, b)$$
$$a \downarrow b = min(a, b).$$

These operations have properties similar to addition and multiplication, respectively. We will refer to the minimum of a set of values as the *product*, and the maximum of a set of values as the *sum*. Signal strengths can be described by an algebraic system $(\mathfrak{T}, \uparrow, \downarrow, 0, \gamma_p)$ which satisfies the following (not exhaustive) list of properties:

1a. $(\mathfrak{T}, \uparrow, 0)$ is a monoid:

i.	$a, b \in \mathfrak{T} \Rightarrow a \uparrow b \in \mathfrak{T}$	(closed)
ii.	$a \uparrow (b \uparrow c) = (a \uparrow b) \uparrow c$	(associative)
iii.	$a \uparrow 0 = 0 \uparrow a = a$	(0 is the identity)

b. $(\mathfrak{T}, \downarrow, \gamma_p)$ is a monoid:

i.	$a, b \in \mathfrak{T} \Rightarrow a \downarrow b \in \mathfrak{T}$	(closed)
ii.	$a \downarrow (b \downarrow c) = (a \downarrow b) \downarrow c$	(associative)
iii.	$a \downarrow \gamma_p = \gamma_p \downarrow a = a$	(γ_p is the identity)

c. 0 is an annihilator for $\downarrow$:

$$a \downarrow 0 = 0$$

2. $\uparrow$ is commutative and idempotent:

i.	$a \uparrow b = b \uparrow a$
ii.	$a \uparrow a = a$

3. $\downarrow$ distributes over $\uparrow$:

$$a \downarrow (b \uparrow c) = (a \downarrow b) \uparrow (a \downarrow c).$$

4. If $a_1, a_2, \ldots, a_p, \ldots$ is a countable sequence of elements in $\mathfrak{T}$, then $a_1 \uparrow a_2 \uparrow \ldots \uparrow a_i \uparrow \ldots$ exists and is unique. Moreover, associativity, commutativity, and idempotence apply to infinite as well as finite sums (sequences of values combined with $\uparrow$).

5. $\downarrow$ distributes over countably infinite sums as well as finite ones.

The above properties indicate that the strength algebra forms a *closed semiring* as described in Aho, Hopcroft, and Ullman [1]. It does not form a ring however, because most elements lack inverses under the sum operation $\uparrow$. Furthermore, the sum operation for a ring need not be idempotent.

We can define vectors and matrices of strength values, which will allow us to describe the network structure and signal strengths in terms of matrix equations. If the pointwise extension of $\uparrow$ (1) is applied to two vectors, the resulting vector will have elements equal to the componentwise maximum (minimum) of the two vectors. The pointwise extension of $\uparrow$ then has properties similar to vector and matrix addition. We can define a matrix product $\cdot$ for strength values where $\uparrow$ is analogous to addition and $\downarrow$ is analogous to multiplication. If $A = B \cdot C$, then

$$a_{ij} = \uparrow_k (b_{ik} \downarrow c_{kj}). \tag{5.1}$$

The properties of closed semirings also hold for the algebra when extended to square matrices. That is $(\mathcal{S}^{n\times n}, \uparrow, \cdot, 0, I)$ also forms a closed semiring, where $\mathcal{S}^{n\times n}$ denotes the set of $n\times n$ matrices over $\mathcal{S}$. 0 denotes the matrix whose elements are all 0, and I denotes the matrix with γ_p's on the diagonal and 0's elsewhere.

For any closed semiring, the *closure* operator, denoted *, is defined to give the reflexive, transitive closure of its argument. For example, if $A \in \mathcal{S}^{n\times n}$,

$$A^* = I \uparrow A \uparrow A\cdot A \uparrow A\cdot A\cdot A \uparrow \ldots = \uparrow_{0\le k<\infty} A^k. \tag{5.2}$$

This operator has the property that

$$A^* = I \uparrow A\cdot A^*. \tag{5.3}$$

The closure operator will be useful for analyzing the conductance paths in a network.

The operations $\uparrow$ and $\bullet$ obey some of the properties of closed semirings even when applied to rectangular matrices and to matrices of different dimensions. In a matrix equation, as long as $\uparrow$ is applied only to conformable arguments and $\bullet$ is applied only to arguments with the number of columns in the first equal to the number of rows in the second, then:

1. $\uparrow$ and $\bullet$ are associative
2. $\uparrow$ is commutative and idempotent
3. $\bullet$ distributes over $\uparrow$.

These properties will allow us to manipulate and transform matrix equations in the strength algebra much as in more traditional matrix algebras.

The set $\mathfrak{I}$ with the ordering $\leq$ form a complete lattice [35] with $\uparrow$ and $\downarrow$ serving as the least upper bound and greatest lower bound operations. Hence the set of vectors $\mathfrak{I}^n$ with the extension of $\leq$ to vectors also forms a complete lattice. For this lattice the pointwise extensions of $\uparrow$ and $\downarrow$ serve as the least upper bound and greatest lower bound operations, respectively. As a consequence, both $\downarrow$ and $\uparrow$ must be monotonic functions, and therefore * must be as well. Only the most elementary aspects of lattice theory will be used in this presentation.

One further function over strength values will be required to express the ability of a stronger signal to block a weaker when both signals are described by their strength values. This will prove important when the equations in the algebra of logic signals are factored into equations in the algebra of signal strengths, as will be described later. Define the function *block* : $\mathfrak{I}\times\mathfrak{I}\rightarrow\mathfrak{I}$ as follows

$$block(a, b) = \begin{cases} a, & a \geq b \\ 0, & a < b \end{cases} \qquad (5.4)$$

We can apply the pointwise extension of this function to vector arguments. The function *block* is *monotonic* in its first argument and *antimonotonic* in its second. That is, if $a \leq b$ then for any c

$$block(a, c) \leq block(b, c)$$

$$block(c, a) \geq block(c, b).$$

As a result, it satisfies the following properties with respect to the least upper bound operation $\uparrow$:

$$block(a \uparrow b, c) = block(a, c) \uparrow block(b, c)$$
$$block(block(a, b), c) = block(a, b \uparrow c)$$
$$b \uparrow block(a, b) = b \uparrow a.$$

Furthermore, 0 serves as an identity element for the second argument:

$$block(a, 0) = a,$$

and *block* is idempotent

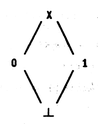
$$block(a, a) = a.$$

This list of properties is by no means exhaustive. In fact all of the above properties hold for the identity function of the first argument. However, they will allow us to manipulate and solve equations involving the function *block*.

5.4 The Algebra of Signal States

The network model allows node states in the set { 0, 1, X }, with 0 and 1 representing the Boolean logic states and with X representing an undefined or erroneous state. We have also introduced a new value $\perp$ representing a null state. This value will only be associated with the *null* signal which has strength 0, indicating that this signal is devoid of state.

The set of signal states is denoted $\mathbb{T} = \{ \perp, 0, 1, X \}$ and the elements are partially ordered as follows:



The least upper bound operation for this partial ordering, denoted $\sqcup$, has the following function table

$\sqcup$	$\perp$	0	1	X
$\perp$	$\perp$	0	1	X
0	0	0	X	X
1	1	X	1	X
X	X	X	X	X

The rule for combining logic signals of equal strength is that the resulting signal will have state equal to the original states if they were equal and state X if they were not. Assuming the value $\perp$ is only associated with signals of strength 0, this rule is expressed by the operation $\sqcup$. That is, if two signals with equal strength and states x and y are combined, the resulting signal will have state $x \sqcup y$. The operation $\sqcup$ is termed the *consistency* operation, because for nonnull arguments it gives a "proper" value (0 or 1) only if the arguments are both proper and equal. Otherwise it gives an error value X.

The set $\mathbb{T}$ with the ordering $\leq$ forms a lattice. Hence the operation $\sqcup$ obeys the following properties:

1. $y \sqcup y = y$ (idempotency)
2. $x \sqcup y = y \sqcup x$ (commutativity)
3. $z \sqcup (x \sqcup y) = (z \sqcup x) \sqcup y$ (associativity)
4. $x \leq y \Rightarrow z \sqcup x \leq z \sqcup y$ (monotonicity)

The set of signal state vectors $\mathbf{v}^n$ along with the extension of $\leq$ to vectors also forms a lattice with the pointwise extension of $\sqcup$ serving as the least upper bound operation. Hence, the pointwise extension of $\sqcup$ also satisfies the properties listed above.

The lattice $\langle V, \leq \rangle$ has both the structure and interpretation of the flat lattices used in denotational semantics of programming languages [35, 49]. The "proper" values 0 and 1 are incomparable, while the bottom element $\perp$ represents an "undefined" or null value and the top element x represents an "overdefined" or erroneous value.

5.5 The Algebra of Signals

5.5.1 Signal Values

A logic signal is represented by a pair of values $\langle s, y \rangle$ where $s \in T$ is the strength and $y \in V$ is the state with the restriction that $y = \perp$ if and only if $s = 0$. That is, only a null signal can have a null state.

Logic signals form a set

$$\Lambda = \{s_1, \ldots, s_q\} \times \{y_1, \ldots, y_p\} \times \{0,1,x\} \cup \{\langle 0, \perp \rangle\}.$$

The null signal $\langle 0, \perp \rangle$ is denoted λ. The expressions $+s$, $-s$, and xs denote signals with strength s and states 1, 0, and X, respectively. The symbols $+$, $-$, and x can be viewed as denoting unary functions mapping strength values to signals of a particular state with the convention that $+0 = -0 = x0 = \lambda$. These functions can be extended pointwise to vectors and matrices as well.

Signal-valued variables are written with italicized characters, while strength and state-valued variables are written with normal characters.

Let $\langle s \rangle$ denote the state of signal s and R of s denote its strength. These symbols denote functions $\langle \cdot \rangle : \mathcal{V} \to \mathcal{T}$ and R of $s : \mathcal{V} \to \mathcal{Y}$.

5.5.2 Signal Combination

The binary operation $\vee$ is defined to describe the effect of combining two signals. This operation is defined in terms of the strength and state of the resulting signal:

$$R \text{ of } (s \vee t) = R \text{ of } s \oplus R \text{ of } t$$

$$\langle s \vee t \rangle = \begin{cases} \langle s \rangle, & R \text{ of } s > R \text{ of } t \\ \langle s \rangle, & R \text{ of } s < R \text{ of } t \\ \langle s \rangle \sqcup \langle t \rangle, & R \text{ of } s = R \text{ of } t. \end{cases}$$

This operation provides a formal statement of the rules that a stronger signal will override a weaker, and that signals of equal strength will combine to form a signal with the same strength and with state equal to the least upper bound of the two states. Note that this operation has only a distant relation to the Boolean "or" operation which is sometimes denoted $\vee$.

The operation $\vee$ defines the following partial ordering among signal values:

That is $a \leq b$ if and only if $a \vee b = b$. With this partial ordering we must maintain a distinction between the terms "greater" and "stronger". The signal ^{x}s is greater than ^{-}s or ^{+}s but not stronger.

The set of signal values $\mathcal{A}$ with the partial ordering $\leq$ forms a lattice with minimum element λ, maximum element $x\gamma_p$, and with $\vee$ serving as the least upper bound operation. As a consequence, $\vee$ must be idempotent, commutative, associative, and monotonic.

We have defined partial orderings for the sets of signal strengths $\mathcal{S}$, signal states $\mathcal{T}$, and signals $\mathcal{A}$. In many cases the functions from one domain to another preserve these orderings, i.e. they are monotonic. For example, the functions +, -, and x are monotonic, because signals with the same state are totally ordered by their strengths. Similarly the function $\|\cdot\|$ is monotonic because signals of different strengths are totally ordered by their strengths. The function $\langle\cdot\rangle$, on the other hand, is *not* monotonic. For example $-\gamma_1 \leq +\gamma_2$, but $\langle -\gamma_1 \rangle \not\leq \langle +\gamma_2 \rangle$.

The pointwise extensions of $\langle\cdot\rangle$, $\|\cdot\|$, and $\vee$ can be defined and obey all of the properties listed thus far.

5.5.3 Signal Factorization

As shall be seen, the algebra of logic signals lacks many of the properties one might desire in a mathematical system, such as a total ordering and monotonicity of certain operations. As a consequence, we will often factor equations in the algebra of signals into equations in the more tractable algebra of signal strengths to aid the mathematical development.

The domain of logic signals loosely resembles complex numbers with strength corresponding to magnitude and state corresponding to phase. Just as a complex number is characterized by either its magnitude and phase or by its real and imaginary parts, a signal is characterized by either its strength and state or by its "1" and "0" parts. The *factored* form of a signal is a pair of strength values (u, d), with u indicating the strength with which the signal will pull a node toward 1, and d indicating the strength with which the signal will pull a node toward 0. The following table shows the two representations of a signal

Signal	Strength - State	1 part - 0 part
λ	⟨0, ⊥⟩	(0, 0)
+s	⟨s, 1⟩	(s, 0)
−s	⟨s, 0⟩	(0, s)
xs	⟨s, X⟩	(s, s)

The factored form must either have both parts equal or one part equal to zero, corresponding to the rule that a stronger signal will override a weaker, and the weaker can be ignored. A signal with state X has equal 1 and 0 parts, because X represents a conflict between signals of equal strength pulling a node toward 1 and 0. The null signal λ, on the other hand, has no ability to pull a node toward 1 or 0: Observe that the ordering $\leq$ between signals is equivalent to the extension of the strength ordering $\leq$ to the factored form representation. The functions $\lceil\cdot\rceil$ and $\lfloor\cdot\rfloor$ are defined to select the 1 and 0 parts of a signal, respectively.

$$\lceil a \rceil = \begin{cases} \|a\|, & \langle a \rangle = 1 \text{ or } X \\ 0, & \langle a \rangle = 0 \text{ or } \bot \end{cases} \tag{5.5}$$

$$\lfloor a \rfloor = \begin{cases} \|a\|, & \langle a \rangle = 0 \text{ or } X \\ 0, & \langle a \rangle = 1 \text{ or } \bot \end{cases} \tag{5.6}$$

The factored form of a signal can be viewed as describing two signals, one with state 1 (or ⊥) and one with state 0 (or ⊥), which when combined will yield the original signal:

$$a = +\lceil a \rceil \vee -\lfloor a \rfloor \tag{5.7}$$

The strength of a signal equals the maximum of its two parts:

$$\|a\| = \lceil a \rceil \uparrow \lfloor a \rfloor \tag{5.8}$$

The state of a signal can also be determined by combining the states of the signals represented by its 1 and 0 parts:

$$\langle a \rangle = \langle +\lceil a \rceil \rangle \sqcup \langle -\lfloor a \rfloor \rangle. \tag{5.9}$$

This identity holds, because at least one part of a signal will equal zero unless the signal state equals X.

For example

$$\langle\!\langle +\ulcorner +\gamma_1 \urcorner \rangle\!\rangle \;=\; \langle\!\langle +\gamma_1 \rangle\!\rangle \sqcup \langle\!\langle -0 \rangle\!\rangle \;=\; \langle\!\langle +\gamma_1 \rangle\!\rangle \sqcup \langle\!\langle \lambda \rangle\!\rangle \;=\; 1 \sqcup \bot \;=\; 1.$$

When two signals are combined with the operation $\vee$, the resulting signal has the following factored form:

$$\begin{aligned} \ulcorner a \vee b \urcorner &= \mathit{block}(\ulcorner a \urcorner \uparrow \ulcorner b \urcorner, \| a \vee b \|) \\ \llcorner a \vee b \lrcorner &= \mathit{block}(\llcorner a \lrcorner \uparrow \llcorner b \lrcorner, \| a \vee b \|) \end{aligned} \tag{5.10}$$

In the above equations the function *block* preserves the restriction that if one part of a signal is less than the other (or equivalently, it is less than the maximum of the two parts), it must equal zero. For example

$$\begin{aligned} \ulcorner +\gamma_2 \vee -\gamma_1 \urcorner &= \mathit{block}(\gamma_2 \uparrow 0, \gamma_2) = \gamma_2 \\ \llcorner +\gamma_2 \vee -\gamma_1 \lrcorner &= \mathit{block}(0 \uparrow \gamma_1, \gamma_2) = 0. \end{aligned}$$

These identities illustrate how the function *block* enters when equations in the signal algebra are factored into equations in the strength algebra.

5.5.4 Signal Coupling

To complete the set of operations for manipulating signal values we require a means of expressing the effect of a signal coupled through a logical conductance. Our rule for signal coupling is that a signal a will be coupled through a logical conductance of nonzero strength s to form a signal with strength $\| a \| \downarrow s$ and state $\langle\!\langle a \rangle\!\rangle$. In Chapter 4 we defined the function $cple: \mathcal{A} \times \mathcal{S} \rightarrow \mathcal{A}$ to yield the signal resulting from an application of this rule. For our formal development, instead of using a function with different types of arguments, we will express signal coupling with the binary operation $\circ$ over signal values. The conductance s is then represented by the signal xs, indicating that it can conduct 0 and 1 signals equally well. Define $\circ$ as follows:

$$a \circ b \;=\; +(\overline{a}\downarrow f\overline{b}) \vee -(\underline{a}\downarrow \underline{b}). \tag{5.11}$$

In other words, if two signals are represented in factored form, $\circ$ is equivalent to the pointwise extension of $\downarrow$ applied to the corresponding parts of the two signals. The signal a coupled through a logical conductance s then forms a signal $xs \circ a$. In all cases used here, one argument of $\circ$ will have state X (or $\perp$.) This formulation takes advantage of the fact that if we let λ_X denote the set:

$$\lambda_X \;=\; \{xs \,|\, s \in \mathcal{Y}\},$$

then the subalgebra $(\lambda_X, \vee, \circ, \lambda, x\gamma_p)$ is isomorphic to the algebra $(\mathcal{Y}, \top, \downarrow, 0, \gamma_p)$. Thus when the logical conductance s is represented by the signal xs, it has the same algebraic properties.

The operation $\circ$ obeys the following properties:

1. $(\lambda, \circ, x\gamma_p)$ is a monoid:
   - i. $a, b \in \lambda \Rightarrow a \circ b \in \lambda$ (closed)
   - ii. $a \circ (b \circ c) = (a \circ b) \circ c$ (associativity)
   - iii. $a \circ x\gamma_p = x\gamma_p \circ a = a$ ($x\gamma_p$ is the identity)
2. λ is an annihilator for $\circ$:
   $a \circ \lambda = \lambda$.

Thus the algebraic system $(\lambda, \vee, \circ, \lambda, x\gamma_p)$ *almost* obeys all of the properties of a closed semiring. In general, however, $\circ$ is *not monotonic*. For example $+\gamma_1 \leq -\gamma_2$, but

$$x\gamma_1 \circ +\gamma_1 \;=\; +\gamma_1 \;\not\leq\; -\gamma_1 \;=\; x\gamma_1 \circ -\gamma_2.$$

As a consequence, $\circ$ *does not distribute over* $\vee$.[1] For example

1. Any operation which distributes over the least-upper bound operation for a lattice must be monotonic. For example, suppose $a \leq c$. If it were the case that $\circ$ distributes over $\vee$ then

$$b \circ a \vee b \circ c \;=\; b \circ (a \vee c) \;=\; b \circ c$$

which implies that $b \circ a \leq b \circ c$. The converse statement need not hold, unless the set is totally ordered.

$$x\gamma_1 \circ (+\gamma_1 \vee -\gamma_2) = -\gamma_1 \not\leq x\gamma_1 \circ +\gamma_1 \vee x\gamma_1 \circ -\gamma_2 = x\gamma_1. \tag{5.12}$$

This lack of distributivity expresses in mathematical terms that signal paths cannot be analyzed independently, because weaker signals may be blocked along a path by a stronger signal with a different state. This phenomenon was demonstrated with the networks shown in Figure 4.7. In fact the left hand side of equation 5.12 expresses how the steady state signal on node n_2 is formed in the first example of Figure 4.7, while the right hand side expresses how the steady signal on node n_2 is formed in the second example. This lack of distributivity will cause some difficulty in our mathematical development.

If we restrict our attention to strong transistors in the 1 and 0 state, however, distributivity (and hence monotonicity) holds. That is, if $b \in \{ \lambda, x\gamma_p \}$ then

$$b \circ (a \vee c) = b \circ a \vee b \circ c,$$

and hence the algebraic system $(\{ \lambda, x\gamma_p \}, \vee, \circ, \lambda, x\gamma_p)$ does form a closed semiring. In fact, this algebraic system is equivalent to the Boolean closed semiring $(\{ 0, 1 \}, +, \cdot, 0, 1)$. This shows that restricted logical conductance networks approach relay networks in their simplicity.

If one of the arguments to $\circ$ represents a conductance value, then $\circ$ is monotonic in this argument. That is, if $b, c \in \mathcal{T}$, and $b \leq c$, then

$$xb \circ a \leq xc \circ a$$

for any $a \in \mathcal{A}$.

The operation $\circ$ was seen to equal to pointwise extension of $\downarrow$ to the factored representation of signals. This leads to the following identities:

$$\begin{aligned} \lceil a \circ b \rceil &= \lceil a \rceil \downarrow \lceil b \rceil \\ \lfloor a \circ b \rfloor &= \lfloor a \rfloor \downarrow \lfloor b \rfloor \\ \| a \circ b \| &= (\lceil a \rceil \downarrow \lceil b \rceil) \uparrow (\lfloor a \rfloor \downarrow \lfloor b \rfloor). \end{aligned} \tag{5.13}$$

For the common case where $\ll a \gg$ or $\ll b \gg$ is in the set $\{ X, \perp \}$:

$$\|a \circ b\| = \|a\| \downarrow \|b\| \tag{5.14}$$

5.5.5 Matrix Operations

The matrix product operation ∘ is defined to describe the effects of first coupling a set of signals through logical conductances and then combining them. That is, if $c = A \circ b$ then

$$c_i = \bigvee_j (a_{ij} * b_j). \tag{5.15}$$

The operation ∘ is closed and associative. Therefore the algebraic system $(\mathcal{A}^{n \times n}, \vee, \circ, 0, I)$ *almost* obeys the properties of a closed semiring, where the identity matrix I has $x\gamma_p$'s on the diagonal and λ's elsewhere, while the zero matrix 0 consists of all λ's. In general, however, ∘ does not distribute over $\vee$ and hence is not monotonic.

In the special case where $A \in \{\lambda, x\gamma_p\}^{n \times m}$, though,

$$A \circ (b \vee c) = A \circ b \vee A \circ c,$$

and for this restricted case ∘ is monotonic. The algebraic system $(\{\lambda, x\gamma_p\}^{n \times n}, \vee, \circ, 0, I)$ then forms a closed semiring. Therefore we can define the closure operation * for this restricted domain as:

$$A^* = I \vee A \vee A \circ A \vee A \circ A \circ A \vee \ldots = \bigvee_{0 \le k < \infty} A^k. \tag{5.16}$$

This closure is isomorphic to the Boolean transitive closure.

If one of the arguments to ∘ represents a conductance matrix, then ∘ is monotonic for this argument. That is if $B, C \in \mathcal{F}^{m \times n}$, and $B \le C$, then

$$xB \circ a \le xC \circ a \tag{5.17}$$

for any $a \in \mathcal{A}^n$.

The following identity follows from the properties of $\vee$ and $\circ$.

$$\| A \circ b \| = (\lceil A \rceil \circ \lceil b \rceil) \uparrow (\lfloor A \rfloor \circ \lfloor b \rfloor). \tag{5.18}$$

For the common case where each element of A is either a null signal or has state X, i.e. $A \in \mathcal{A}_X{}^{n \times m}$,

$$\| A \circ b \| = \| A \| \circ \| b \|. \tag{5.19}$$

A matrix product can also be factored into its 1 and 0 parts:

$$\begin{aligned} \lceil A \circ b \rceil &= block(\lceil A \rceil \circ \lceil b \rceil, \| A \circ b \|) \\ \lfloor A \circ b \rfloor &= block(\lfloor A \rfloor \circ \lfloor b \rfloor, \| A \circ b \|) \end{aligned} \tag{5.20}$$

The pointwise extension of the function *block* maintains the restriction that each part of a signal must equal the strength of the signal or equal 0.

5.6 Summary

The concept and properties of logic signals have been formalized into an abstract algebra with a domain corresponding to the signal values and with operations corresponding to the rules for combining and coupling signals. Many of the properties of logic signals are reflected by this algebra: the domain is a small, discrete set, and the operations obey many desirable mathematical properties. Even the complications arising from signal blocking are reflected by the lack of distributivity in the algebra except for a restricted domain.

The domains and their operations are summarized below.

Signal Strengths

Elements: $\mathcal{S} = \{ 0, \kappa_1, \ldots, \kappa_q, \gamma_1, \ldots, \gamma_p \}$

Ordering: $0 < \kappa_1 < \ldots < \kappa_q < \gamma_1 < \ldots < \gamma_p$

Operations:

$\uparrow$	maximum (least upper bound)
$\downarrow$	minimum (greatest lower bound)
block	signal blocking
$\cdot$	$(\uparrow \downarrow)$ matrix product
*	closure

Signal States

Elements: $\mathcal{V} = \{ \perp, 0, 1, X \}$

Ordering:

Operations:

$\sqcup$	consistency (least upper bound)

Signals

Elements: $\mathcal{A} = \{\lambda, {}^{+}\kappa_1, {}^{-}\kappa_1, {}^{\times}\kappa_1, \ldots, {}^{+}\gamma_p, {}^{-}\gamma_p, {}^{\times}\gamma_p\}$

Ordering:

Operations:

$\vee$	signal combination (least upper bound)
$\circ$	signal coupling
o	($\vee$ $\circ$) matrix product
*	closure (for restricted domain only)

Functions from signals to states:

$\ll\cdot\gg$	state of signal

Functions from signals to strengths:

$\|\|\cdot\|\|$	strength of signal
$\lceil\cdot\rceil$	1 part
$\lfloor\cdot\rfloor$	0 part

Functions from strengths to signals:

+	form signal with state 1 (or $\perp$)
-	form signal with state 0 (or $\perp$)
x	form signal with state X (or $\perp$)

6. Computation of the Target State

6.1 Introduction

The algebra of logic signals along with the related algebras of signal strengths and signal states provide a set of mathematical tools for further developing the switch-level abstraction. In this chapter a method will be developed for finding the minimum solution of the steady state signal equation given in Chapter 4, which then gives us a method for finding the steady state of a logical conductance network. This method is then generalized into one for finding the target state of an arbitrary switch-level network. This development will utilize only the logic signal abstraction as expressed by our algebras, along with two equations which were derived from the properties of the electrical model: the definition of the target state in terms of the steady states of a set of logical conductance networks given in equation 3.20, and the equation for the steady state signals given in equation 4.5. While we could arrive at the final results more directly by utilizing additional properties of the electrical model, this approach demonstrates that the concept of logic signals is quite powerful and self-contained. The earlier work with the electrical model was presented only to motivate and justify the more abstract concepts. The issues of implementing these techniques with efficient computer algorithms are deferred to Chapter 7. In particular, the matrix notation in this chapter is used only for mathematical convenience and need not imply that the simulation algorithms involve matrix operations.

6.2 The Target State Equation

First, let us restate the definition of the target state in terms of the steady states of a set of logical conductance networks. The function *target* (x, y, z) was defined as giving the set of states $\tilde{y}$ which the normal nodes would reach if the input nodes were held in state x, the transistors were held in state z, and the normal nodes were initialized to state y. With the transistors held in state z the network of transistors can be described by a set of logical conductance networks, where a transistor in the 1 state forms a logical

conductance equal to its strength, a transistor in the 0 state forms a logical conductance of 0 (i.e. an open circuit), and a transistor in the X state forms a logical conductance either equal to its strength or to 0. We have seen that a set of parallel logical conductances can be replaced by a single logical conductance equal to the maximum element in the set. Thus the range of logical conductance networks corresponding to a switch-level network in transistor state z can be described by four logical conductance matrices: G^{zmn}, G^{zmx}, E^{zmn}, and E^{zmx}. Each element g^{zmn}_{ij} of G^{zmn} equals the maximum strength of all transistors in the 1 state connecting normal nodes n_i and n_j or equals 0 if no such transistor exists. The elements of G^{zmx} equal the corresponding values for all transistors in either the 1 or the X state. Each element e^{zmx}_{ij} of E^{zmx} equals the maximum strength of all transistors in the 1 state connecting normal node n_i and input node i_j or equals 0 if no such transistor exists. The elements of E^{zmx} equal the corresponding values for transistors in the 1 or X state. More formally, if T_{ij} denotes the following set of transistors:

$$T_{ij} = \{ i_k \mid (\text{SOURCE}(i_k)=n_i \text{ and } \text{DRAIN}(i_k)=n_j) \text{ or } (\text{SOURCE}(i_k)=n_j \text{ and } \text{DRAIN}(i_k)=n_i) \},$$

then

$$e^{zmn}_{ij} = \max_{i_k \in \vec{T}_{ij}, z_k = 1} str_k. \tag{6.1}$$

and

$$g^{zmn}_{ij} = \max_{i_k \in \vec{T}_{ij}, z_k \in \{1, x\}} str_k. \tag{6.2}$$

In both equations, the maximum of an empty set is defined to equal 0. Both G^{zmn} and G^{zmx} are symmetric matrices with elements in the set $\{0, \gamma_1, \ldots, \gamma_p\}$. In general these matrices are very sparse, because each node is connected to only a limited number of other nodes. Similarly, if T'_{ij} denotes the following set of transistors:

$$T'_{ij} = \{ i_k \mid (\text{SOURCE}(i_k)=n_i \text{ and } \text{DRAIN}(i_k)=i_j) \text{ or } (\text{SOURCE}(i_k)=i_j \text{ and } \text{DRAIN}(i_k)=n_i) \},$$

then

$$c^{zmn}_{ij} = \max_{i_k \in \vec{T}'_{ij}, z_k = 1} str_k. \tag{6.3}$$

and

$$e^{max}_{ij} = \overset{\uparrow}{\underset{t_k \in T'_{ij},\, z_k \in \{1, X\}}{}} str_k. \qquad (6.4)$$

E^{min} and E^{max} are $n \times m$ matrices with elements in the set $\{0, \gamma_1, \ldots, \gamma_p\}$. These matrices may be less sparse than G^{min} and G^{max}, because some input nodes (e.g. VDD, GND) serve as source and drain nodes for many transistors. The network may also contain transistors connecting input nodes to one another, but these have no effect on the logical behavior. To compute the target state function, we need only look at the network of logical conductances described by these four matrices, without considering the transistor configurations or states which give rise to them.

The presence of transistors in the X state implies that the actual conductance matrices G and E lie within the ranges:

$$\begin{aligned} G^{min} \leq G \leq G^{max} \\ E^{min} \leq E \leq E^{max}. \end{aligned} \qquad (6.5)$$

Let $\{G\}$ and $\{E\}$ denote the following sets of matrices

$$\{E\} = \{E \mid e_{jk} = e^{min}_{jk}(\rho) \text{ or } e_{jk} = e^{max}_{jk}(\rho)\}$$

$$\{G\} = \{G \mid g_{jk} = g^{min}_{jk}(\rho) \text{ or } g_{jk} = g^{max}_{jk}(\rho), \text{ and } g_{jk} = g_{kj}\}.$$

Note that these definitions of $\{E\}$ and $\{G\}$ are equivalent to those given in Chapter 4 in terms of the sets of order of magnitude network matrices E and G as were defined in equations 3.7 and 3.8.

For logical conductance matrices G and E, let $\bar{y}(G, E)$ denote the steady state of the logical conductance network with these values of logical conductances. Equation 3.20 defines the target state of a node in the switch-level network as

$$\tilde{y}_i = \begin{cases} 1, & \bar{y}_i(G, E) = 1 \text{ for all } G \in \{G\} \text{ and } E \in \{E\} \\ 0, & \bar{y}_i(G, E) = 0 \text{ for all } G \in \{G\} \text{ and } E \in \{E\} \\ X, & \text{else.} \end{cases} \qquad (3.20)$$

That is the target state equals 1 or 0 if and only if it has this unique state regardless of the conductances formed by transistors in the X state, and otherwise it equals X. This equation can be expressed more concisely using the consistency operation ⊔ for logic states:

$$\vec{y} = \bigsqcup_{\{G\},\{E\}} \vec{y}(G, E). \tag{6.6}$$

This equation shows that the target state function for an arbitrary switch-level network can be determined by computing the steady states for the logical conductance networks represented by all possible matrices G and E in the sets { G } and { E } and then combining these states with the operation ⊔. This reduces the problem of computing the target state equation for a switch-level network to one of computing the steady state of a logical conductance network. As shall be seen later, the method for computing the steady state of a logical conductance network can be generalized into a method for directly computing the target state of a switch-level network.

6.3 The Steady State Signal Equation

Let us turn our attention to computing the steady state $\vec{y}$ for a particular set of conductance matrices G and E.

The rule for forming logic signals is that an input node forms a logic signal with state equal to the node state and strength equal to γ_p. As was defined in Chapter 4, the vector x denotes the set of signals formed by the input nodes in state x:

$$\langle x_i \rangle = x_i \tag{6.7}$$
$$\| x_i \| = \gamma_p$$

Similarly, a normal node forms an initial signal with state equal to the node state and strength equal to the node size. As was defined in Chapter 4, the vector *y* denotes the set of signals formed by the normal nodes in state y:

$$\langle y_i \rangle = y_i \qquad (6.8)$$
$$\| y_i \| = cap_i.$$

In Chapter 4 it was shown that the vector of steady state signals ν must be the minimum vector satisfying the constraints:

$$\nu_i = \text{l.u.b.}\left(\{ y_i \} \cup \{ cple(x_j, e_{ij}) \mid 1 \le j \le m \} \cup \{ cple(\nu_j, g_{ij}) \mid 1 \le j \le n \} \right), \qquad (4.5)$$

for all i. This equation can be expressed using operations in the algebra of logic signals as

$$\nu_i = y_i \vee \bigvee_j (x e_{ij} \circ x_j) \vee \bigvee_j (\nu g_{ij} \circ \nu_j).$$

If the matrices E and G are defined as the matrices of logic signals representing the logical conductance matrices E and G, i.e.

$$E = xE \qquad (6.9)$$
$$G = xG$$

then this set of equations can be expressed by a single matrix equation:

$$\nu = E \circ x \vee y \vee G \circ \nu. \qquad (6.10)$$

Unlike equations in other algebras, in which all expressions involving the dependent variables can be move to one side of the equality, we have no inverse operation in the signal algebra to permit cancellation of the left hand side. Hence the equation for the steady state signal must be expressed as a recurrence relation. It will be shown shortly that equation 6.10 has a unique minimum solution.

6.4 Solution for Restricted Logical Conductance Networks

We wish to find the minimum vector v such that $v = f(v)$, where

$$f(a) = E \circ x \vee y \vee G \circ a.$$

A general technique for solving such recurrence equations is only known for *monotonic* recurrence equations, i.e. ones in which the recurrence is expressed by a monotonic function. In general, the nonmonotonicity of the operation $\circ$ implies that the function f may not be monotonic, and hence this technique does not apply. For restricted logical conductance networks, however, in which normal nodes may only be interconnected by conductances of strength γ_p, the function f is monotonic. That is, a restricted logical conductance network has a conductance matrix G with each element equal to 0 or γ_p. In this case each element of G equals λ or $x\gamma_p$. This implies that for any a and b

$$G \circ (a \vee b) = G \circ a \vee G \circ b,$$

and therefore f is monotonic. The following theorem, a special case of one given by Scott [35], shows how to solve such an equation.

Theorem 6.1.
For a monotonic function $f: \mathcal{A}^n \rightarrow \mathcal{A}^n$, the equation

$$a = f(a)$$

has a unique minimum solution given by

$$a^{min} = \lim_{k \rightarrow \infty} f^k(\theta) \tag{6.11}$$

where θ denotes a vector of all λ's, and the superscript k denotes k applications of the function f. Furthermore, this limit will be reached for some $k \leq n \cdot |\mathcal{A}|$, where $|\cdot|$ denotes set size.

Proof of Theorem 6.1:

Consider the following sequence

$$0,\ f(0),\ f(f(0)),\ \ldots,\ f^k(0),\ \ldots$$

First we will prove that its limit exists. Clearly $0 \leq f(0)$, and by the monotonicity of f one can prove by induction on k that

$$f^k(0) \leq f^{k+1}(0).$$

Hence the sequence is nondecreasing. Any strictly increasing sequence in $\mathcal{A}^n$ would have a length of at most $n \cdot |\mathcal{A}|$. Therefore for some $j \leq n \cdot |\mathcal{A}|$, $f^j(0) = f^{j+1}(0)$, and then for any $k \geq j$, $f^k(0) = f^k(0)$. From this we can see that a^{min} must be a solution, because

$$a^{min} = f^j(0) = f^{j+1}(0) = f(f^j(0)) = f(a^{min}).$$

Finally, suppose for some a, $a = f(a)$. Starting with the basis $a \geq 0$, we can prove by induction on k that

$$a = f^k(a) \geq f^k(0).$$

Therefore

$$a \geq \lim_{k \to \infty} f^k(0) = a^{min}.$$

Thus a^{min} is the unique minimum solution of the recurrence equation.∎

This theorem follows as a special case of a theorem proved by Scott [35] regarding the least fixed point of a continuous function on a continuous lattice, where "continuity" here is only distantly related to the continuity of real analysis. In finding the minimum solution of the recurrence equation $a = f(a)$ we are computing the least fixed point of the function f. Any finite lattice is continuous, as is any monotonic function on a finite lattice. Scott shows that a result similar to equation 6.11 holds for any continuous function on a continuous lattice. Finite convergence, however, may not be guaranteed for functions on infinite lattices.

The process outlined in this theorem of computing the sequence $f(0)$, $f(f(0))$, ..., $f^k(0)$, ... until it converges corresponds to a straightforward *relaxation* algorithm. It starts by setting a_i to λ for each node and then performing relaxations of the form $a_i \leftarrow f_i(a)$ until it converges, i.e. any further relaxation steps would not change a. Since we have expressed the method in vector form, each application of f corresponds to applying relaxation computations at all nodes simultaneously.

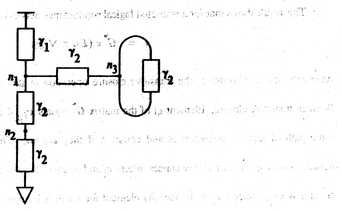
For example the logical conductance network shown in Figure 6.1 models an nMOS Nand gate and pass transistor with all transistors in the 1 state. The extra self-loop has been added to n_3 to introduce the possibility of extraneous solutions. Equation 6.10 can be expanded as

$$\begin{aligned} v_1 &= {}^+\gamma_1 \vee x\gamma_2 \circ v_2 \vee x\gamma_2 \circ v_3 \\ v_2 &= {}^-\gamma_2 \vee x\gamma_2 \circ v_1 \\ v_3 &= \qquad x\gamma_2 \circ v_1 \vee x\gamma_2 \circ v_3. \end{aligned}$$

The above equations have the charging signals y left out for simplicity. The relaxation method gives the following sequences:

$$\begin{array}{lccccccc} a_1: & \lambda & {}^+\gamma_1 & {}^-\gamma_2 & {}^-\gamma_2 & {}^-\gamma_2 & {}^-\gamma_2 & \cdots \\ a_2: & \lambda & {}^-\gamma_2 & {}^-\gamma_2 & {}^-\gamma_2 & {}^-\gamma_2 & \cdots & \\ a_3: & \lambda & \lambda & {}^+\gamma_1 & {}^-\gamma_2 & {}^-\gamma_2 & \cdots & \end{array}$$

Fig. 6.1. Restricted Logical Conductance Network Example.



indicating that all nodes have steady state signals $\neg\gamma_2$ and hence have steady states 0. Observe that an extraneous solution never arises because at all times each value a_i is less than or equal to the steady state signal.

For this particular equation the minimum solution can be expressed in terms of the closure operation.

Corollary 6.1.1.
For a matrix $G \in \{\lambda, x\gamma_p\}^{n\times n}$, and a vector $b \in \mathcal{A}^n$, the equation

$$a = b \vee G \circ a \tag{6.12}$$

has a unique minimum solution given by

$$a^{min} = G^* \circ b \tag{6.13}$$

Proof of Corollary 6.1.1:

The function $f(a) = b \vee G \circ a$ is monotonic, and an expansion of $f^k(\theta)$ gives

$$f^k(\theta) = b \vee G \circ (\ldots b \vee G \circ (b \vee G \circ \theta)\ldots) = \bigvee_{0\le j\le k} G^j \circ b = \left[\bigvee_{0\le j\le k} G^j\right] \circ b$$

Hence

$$a^{min} = \lim_{k\to\infty} f^k(\theta) = G^* \circ b \qquad \blacksquare$$

This result shows that for a restricted logical conductance network the steady state signal is given by

$$s = G^* \circ (E \circ x \vee y). \tag{6.14}$$

As mentioned in Chapter 5, the transitive closure operation in this restricted case is equivalent to a Boolean transitive closure. Element i,j of the matrix G^* equals $x\gamma_p$ if nodes n_i and n_j are connected by some path of logical conductances and equals λ if they are not. Thus the matrix G^* partitions the network into a set of equivalence classes, where n_i and n_j are in the same class if and only if element i,j of G^* equals $x\gamma_p$. Since $x\gamma_p$ is the identity element for $\circ$ and λ is an annihilator, computing element i of

$G^* \circ b$ in this case simply involves selecting the elements of b for the nodes in the same class as n_i and combining them with the operation $\vee$. Thus if we let b equal the vector of signals given by $E \circ x \vee y$, and node n_i is in equivalence class C_k, then

$$n_i = \bigvee_{n_j \in C_k} b_j.$$

Observe that all nodes in a class will have the same steady state signal. If we were only concerned with restricted switch-level networks with no transistors in the X state, the computation of the target state would be quite simple.

6.5 Solution for General Logical Conductance Networks

Unfortunately, the technique outlined for computing the steady state signal in restricted logical conductance networks can fail for general networks, because it may converge on some solution other than the minimum. Consider how the proof of Theorem 6.1 relies on the monotonicity of the function f. First, it is used to show that the sequence $f^k(\theta)$ is monotonic and hence converges. In fact, it appears that this sequence would converge for equations of the form of 6.12 regardless of the monotonicity of f, although this has not been proved formally. More importantly, however, the monotonicity of f is used to show that $a \geq f^k(\theta)$ for any a which satisfies the recurrence relation, and hence the limit to this sequence must be the minimum solution. This result does not hold when the recurrence function f is not monotonic.

The network shown in Figure 6.2 resembles the network shown in Figure 6.1 except that the pass transistor has strength γ_1, and therefore we no longer have a restricted logical conductance network. Equation 6.10 can be expanded as:

Fig. 6.2. General Logical Conductance Network Example

$$
\begin{aligned}
n_1 &= +\gamma_1 \vee x\gamma_2 * n_2 \vee x\gamma_1 * n_3 \\
n_2 &= -\gamma_2 \vee x\gamma_2 * n_1 \\
n_3 &= x\gamma_1 * n_1 \vee x\gamma_2 * n_3
\end{aligned}
$$

The above equations have the charging signals γ left out for simplicity. The minimum solution of these equations is $n_1 = -\gamma_2$, $n_2 = -\gamma_2$, and $n_3 = -\gamma_1$, giving a steady state of 0 on all three nodes, just as one would expect. The relaxation method gives the sequences:

$$
\begin{array}{lccccc}
a_1: & \lambda & +\gamma_1 & -\gamma_2 & -\gamma_2 & -\gamma_2 & \cdots \\
a_2: & \lambda & -\gamma_2 & -\gamma_2 & -\gamma_2 & -\gamma_2 & \cdots \\
a_3: & \lambda & \lambda & +\gamma_1 & x\gamma_1 & x\gamma_1 & \cdots
\end{array}
$$

The sequence converges with the signal $x\gamma_1$ on node n_3, which is greater than the minimum value $-\gamma_1$, and therefore finds an extraneous solution with state X on node n_3.

This error arises due to an interplay between the effects of signal blocking (giving a nonmonotonic recurrence function), and the possibility of extraneous solutions of the equation. The network of Figure 6.2 has been contrived to cause this interplay. On the second relaxation step we introduce information about the path from VDD to n_3 in setting a_3 to $+\gamma_1$, and this value is not less than or equal to the steady

state signal $\neg\gamma_1$. Due to the presence of the self-loop at n_3, this information will remain during further relaxation steps. When the third relaxation step introduces information about the path from GND to n_3 into a_3, it combines with the old value of a_3 to give $\kappa\gamma_1$ instead of $\neg\gamma_1$. Thus our relaxation method does not take the effects of signal blocking into account properly and hence may reach an extraneous solution. While this example seems rather contrived, similar effects can occur with more realistic (but larger) networks.

The steady state signal for a general logical conductance network can be found by a method of *conditioned relaxations* which first computes the strength of the steady state signal and then uses this information while computing successive relaxations to prevent a node from being set to a nonnull signal weaker than the steady state signal. For example, suppose for the network shown in Figure 6.2, we could determine that the steady state signals will have strength γ_2 on nodes n_1 and n_2 and will have strength γ_1 on node n_3. In generating the sequences of values on each node, any time our original method would set the node to a signal weaker than the steady state signal, we will instead set it to λ. This gives the following sequences:

$$
\begin{array}{lccccccc}
n_1\text{:} & \lambda & \lambda & \neg\gamma_2 & \neg\gamma_2 & \neg\gamma_2 & \cdots \\
n_2\text{:} & \lambda & \neg\gamma_2 & \neg\gamma_2 & \neg\gamma_2 & \neg\gamma_2 & \cdots \\
n_3\text{:} & \lambda & \lambda & \lambda & \neg\gamma_1 & \neg\gamma_1 & \cdots
\end{array}
$$

The signal $+\gamma_1$ is weaker than the steady state signal for node n_1 and hence the first relaxation computation at this node will give a value λ. As a consequence, the signal $+\gamma_1$ is never propagated to node n_3 and will never create an extraneous signal. This example shows how the conditioned relaxation technique prevents extraneous signals from arising by killing weak signals before they can become extraneous. We will now prove formally that this method produces the correct results.

6.5.1 Factored Equations

The method of conditioned relaxations can be derived formally by factoring the steady state signal equation into its 1 and 0 parts, giving more tractable equations in the algebra of signal strengths. All elements of E and G are null signals or have state X, which implies by equation 5.19 that

$$\|E \circ x\| = \|E\| \cdot \|x\| = E \cdot \|x\|$$
$$\|G \circ \nu\| = \|G\| \cdot \|\nu\| = G \cdot \|\nu\|.$$

Therefore

$$\|\nu\| = E \cdot \|x\| \uparrow \|y\| \uparrow G \cdot \|\nu\|, \tag{6.15}$$

which shows that $\|\nu\|$ satisfies a recurrence relation of the form $\|\nu\| = h(\|\nu\|)$, where the function h is monotonic. Recurrence relations for $\lceil\nu\rceil$ and $\lfloor\nu\rfloor$ can be derived as well:

$$\lceil\nu\rceil = block(E \cdot \lceil x\rceil \uparrow \lceil y\rceil \uparrow G \cdot \lceil\nu\rceil, \|\nu\|) \tag{6.16}$$
$$\lfloor\nu\rfloor = block(E \cdot \lfloor x\rfloor \uparrow \lfloor y\rfloor \uparrow G \cdot \lfloor\nu\rfloor, \|\nu\|). \tag{6.17}$$

Thus, if we could determine the value of $\|\nu\|$, we would have recurrence relations of the form $\lceil\nu\rceil = f_1(\lceil\nu\rceil)$ and $\lfloor\nu\rfloor = f_0(\lfloor\nu\rfloor)$, where both f_1 and f_0 are monotonic functions. We will show later that $\|\nu\|$, $\lceil\nu\rceil$, and $\lfloor\nu\rfloor$ must be the minimum solutions of their respective recurrence equations.

6.5.2 Recurrence Equations in the Strength Algebra

The minimum solution of a monotonic recurrence equation in the strength algebra can be found by a relaxation method similar to the one shown for the signal algebra, as is proved in the following theorem.

Theorem 6.2.
For a monotonic function $f: \mathcal{F}^n \rightarrow \mathcal{F}^n$, the equation

$$\mathbf{a} = f(\mathbf{a})$$

has a unique minimum solution given by

$$\mathbf{a}^{min} = \lim_{k \rightarrow \infty} f^k(\mathbf{0}) \qquad (6.18)$$

where $\mathbf{0}$ denotes a vector of all 0's. Furthermore, this limit will be reached for some $k \leq n \cdot |\mathcal{F}|$.

The proof of this theorem parallels the proof of Theorem 6.1. For equations of the form of 6.15, the minimum solution can be expressed in terms of the closure operation.

Corollary 6.2.1.
For a matrix $G \in \mathcal{F}^{n \times n}$, and a vector $\mathbf{b} \in \mathcal{F}^n$, the equation

$$\mathbf{a} = \mathbf{b} \dagger G \bullet \mathbf{a}$$

has a unique minimum solution given by

$$\mathbf{a}^{min} = G^* \bullet \mathbf{b}.$$

The proof of this corollary parallels the proof of Corollary 6.1.1. Observe that this result holds for strength values in unrestricted as well as restricted networks.

As a conclusion to our study of recurrence equations in the strength algebra, let us look at the relation between solutions of different equations. Define the relation $\leq$ between two functions f and g as $f \leq g$ if and only if $f(\mathbf{a}) \leq g(\mathbf{a})$ for all $\mathbf{a}$. The following theorem shows that this relation will then hold between the minimum solutions of their respective recurrence equations. This theorem will prove valuable in comparing the steady state signals of different logical conductance networks.

Theorem 6.3.
If monotonic functions $f:\mathbf{y}^n \rightarrow \mathbf{y}^n$ and $g:\mathbf{y}^n \rightarrow \mathbf{y}^n$ are ordered $f \leq g$ and $\mathbf{a}^{min}$ and $\mathbf{b}^{min}$ are the minimum solutions to the equations

$$\mathbf{a} = f(\mathbf{a})$$
$$\mathbf{b} = g(\mathbf{b}),$$

respectively, then

$$\mathbf{a}^{min} \leq \mathbf{b}^{min}.$$

Proof of Theorem 6.3:

By induction on k $f^k(\mathbf{0}) \leq g^k(\mathbf{0})$, and therefore

$$\mathbf{a}^{min} = \lim_{k \rightarrow \infty} f^k(\mathbf{0}) \leq \lim_{k \rightarrow \infty} g^k(\mathbf{0}) = \mathbf{b}^{min}.$$

∎

This theorem is also a special case of one given by Scott which states that the least fixed point operator is monotonic when applied to monotonic functions.

6.5.3 Solution Technique

The following theorem shows that the minimum solutions of equations 6.15, 6.16, and 6.17 do indeed lead to the value of the steady state signal.

Theorem 6.4.
For a matrix $G \in \mathcal{J}^{m \times n}$, and a vector $b \in \mathcal{L}^n$, define G as $G = xG$. The unique minimum solution of the equation $a = f(a)$, where

$$f(a) = b \vee Go\,a \tag{6.19}$$

is given by

$$a^{min} = +u^{min} \vee -d^{min},$$

where u^{min} and d^{min} are the minimum solutions of the equations $u = f_1(u)$ and $d = f_0(d)$, respectively, and the functions f_1 and f_0 are defined as:

$$f_1(u) = block\,(T\,b\uparrow 1\,G \circ u,\, r) \tag{6.20}$$

$$f_0(d) = block\,(L\,b\downarrow 1\,G \circ d,\, r), \tag{6.21}$$

and

$$r = G^{\#} \circ b. \tag{6.22}$$

The proof of this theorem is given in Appendix II. It serves mainly to confirm one's intuition but requires proving many subtle points. Primarily it involves showing that the signal vector $+u^{min} \vee -d^{min}$ satisfies the recurrence relation $a = f(a)$. It is then straightforward to prove that it must be the minimum solution.

If we let the vector b in equation 6.19 equal $R \circ x \vee p$, then we can apply Theorem 6.4 to find the steady state signal r. Alternatively, the result of this theorem can be expressed in a manner more suggestive of a sequence of conditioned reactions in the signal algebra. Define the function

$$kill : \mathcal{A} \times \mathcal{J} \mapsto \mathcal{A} \text{ as}$$

$$kill\,(a, b) = +block\,(T\,a\uparrow, b) \vee -block\,(L\,a\downarrow, b). \tag{6.23}$$

In other words

$$kill(a, b) = \begin{cases} a, & \| a \| \geq b \\ \lambda, & \| a \| < b. \end{cases}$$

This function expresses our technique of killing any signal weaker than some strength value. The following corollary to Theorem 6.4 describes the method of conditioned relaxations.

Corollary 6.4.1.
For a matrix $G \in \mathcal{S}^{n \times n}$, and a vector $b \in \mathcal{A}^n$, if G is defined as $G = xG$, then the minimum solution of the equation $a = f(a)$ where

$$f(a) = b \vee G \circ a.$$

is given by

$$a^{min} = \lim_{k \to \infty} f^k(\theta) \tag{6.24}$$

where

$$f'(a) = kill(f(a), r), \tag{6.25}$$

and

$$r = G^* \cdot \| b \|.$$

The proof of Corollary 6.4.1 is given Appendix II. It proceeds by factoring the function f' and showing that

$$f'^k(\theta) = +f_1^k(0) \vee -f_0^k(0).$$

With this, one can easily see that the sequence will converge to a^{min}.

The conditioned relaxation method succeeds where the straightforward relaxation method fails, because the function f' is monotonic over the domain $\{ a \mid a \leq a^{min}$, and $a = block(a, \| a^{min} \|) \}$, whereas the function f may not be. Furthermore, f' is closed over this domain. Thus, when successive values of a are formed by repeated applications of f', we will get a monotonic sequence converging to a^{min}. This result is given as a corollary to the main theorem, because it does not generalize to switch-level networks containing transistors in the X state while the method given in the theorem does.

6.6 The Target State

Returning to the problem of computing the target state $\tilde{y}$ of a switch-level network, where the network may have conductance matrices in the sets { G } and { E }, we saw that the target state could be found by computing all possible steady states $\bar{y}(G, E)$ and then combining these possible states with the consistency operation ⊔. Such an approach, however, would have exponential complexity if a large number of transistors were in state X. Instead, we will derive a method of directly computing $\tilde{y}$ by generalizing the method of Theorem 6.4.

To determine the target state $\tilde{y}_i$ of node n_i we need only find the range of values $\bar{y}_i(G, E)$ can assume. If some setting of G and E can be found which gives 1 or X for $\bar{y}_i(G, E)$, and some other setting can be found which gives 0 or X, then $\tilde{y}_i$ equals X. If, on the other hand, either of these two attempts fails, then $\tilde{y}_i$ equals the state found by the other attempt. Define $v(G, E)$ as the vector of steady state signals for the logical conductance network with conductance matrices G and E. Then $\bar{y}_i(G, E)$ equals 1 or X if and only if $\lceil v_i(G, E) \rceil$ is greater than 0. Similarly, $\bar{y}_i(G, E)$ equals 0 or X if and only if $\lfloor v_i(G, E) \rfloor$ is greater than 0. This suggests that the target state of a node can be found by performing two optimization processes. The first maximizes $\lceil v_i(G, E) \rceil$ for all possible G and E giving a result u^{opt}_i, while the second maximizes $\lfloor v_i(G, E) \rfloor$ giving a result d^{opt}_i. These values can be combined to give the target state:

$$\tilde{y}_i = \begin{cases} 1, & u^{opt}_i > 0 \text{ and } d^{opt}_i = 0 \\ 0, & d^{opt}_i > 0 \text{ and } u^{opt}_i = 0 \\ X, & \text{else.} \end{cases} \tag{6.26}$$

That is, the target state will equal a proper value (0 or 1) if and only if the corresponding optimization process succeeds (obtains a nonzero value), while the other fails. This can be expressed by the following equation:

$$\tilde{y}_i = \langle +u^{opt}_i \rangle \sqcup \langle -d^{opt}_i \rangle. \tag{6.27}$$

At first this optimization approach might not seem to improve on the full enumeration technique. It seems to call for a separate set of optimizations for each node, with each involving the trial of a number of conductance matrices. Fortunately, these difficulties do not arise. Instead, the values $\Gamma_{\gamma}[G, E)\rceil$ will be maximized for all nodes with one particular pair of matrices G and E, and a similar result holds for $L_{\gamma}[G, E)\rfloor$. Furthermore, these values can be computed without ever finding the particular values of G and E which give rise to them. Instead, the vectors π^{opt} and δ^{opt} can be computed directly by slightly modifying equations 6.20 and 6.21, as is shown in the following theorem.

Theorem 6.5.
The target state $\tilde{\gamma}$ of a switch-level network is given by

$$\tilde{\gamma} = \langle +\pi^{opt} \rangle \sqcup \langle -\delta^{opt} \rangle, \tag{6.28}$$

where π^{opt} and δ^{opt} are the minimum solutions of the equations $\pi = g_1(\pi)$ and $\delta = z_0(\delta)$, respectively, and the functions g_1 and z_0 are defined as

$$g_1(\pi) = block(E^{max} \cdot \lceil x \rceil \uparrow \lceil \gamma \rceil \uparrow G^{max} \cdot \pi, r) \tag{6.29}$$

$$z_0(\delta) = block(E^{max} \cdot \lfloor x \rfloor \uparrow \lfloor \gamma \rfloor \uparrow G^{max} \cdot \delta, x), \tag{6.30}$$

and

$$r = G^{max^*}(E^{max} \cdot \| x \| \uparrow \| \gamma \|). \tag{6.31}$$

The full proof of Theorem 6.5 is given in Appendix II. It involves showing that

$$\pi^{opt} = \{G\}, \{E\} \pi^{opt}(G, E), \tag{6.32}$$

where π^{opt} equals the minimum solution of equation 6.29, and $\pi^{opt}(G, E)$ equals the minimum solution of the equation $\pi = f_1(\pi)$ for

$$f_1(u) = block(E \cdot \lceil x \rceil \uparrow \lceil \gamma \rceil \uparrow G \cdot u, G^*(E \cdot \| x \| \uparrow \| \gamma \|)).$$

That is $u^{min}(G, E)$ equals $\lceil \nu(G, E) \rceil$. We can see that u^{opt} must be greater or equal to any $u^{min}(G, E)$ for any G in { G } and E in { E } as follows. For any such G and E, $G^{min} \leq G \leq G^{max}$ and $E^{min} \leq E \leq E^{max}$, and since *block* is monotonic in its first argument and antimonotonic in its second, this implies that $f_1 \leq g_1$. Therefore Theorem 6.3 shows that u^{opt} must be greater than or equal to $u^{min}(G, E)$. To complete the proof, we need only find a matrix $G \in \{ G \}$ and a matrix $E \in \{ E \}$ which give $u^{min}(G, E) = u^{opt}$. The following matrices satisfy this requirement, although the proof is rather tedious:

$$g_{ij} = \begin{cases} g^{min}_{ij}, & u^{opt}_i = 0 \text{ or } u^{opt}_j = 0 \\ g^{max}_{ij}, & u^{opt}_i > 0 \text{ and } u^{opt}_j > 0, \end{cases}$$

$$e_{ij} = \begin{cases} e^{min}_{ij}, & \lceil x_j \rceil = 0 \\ e^{max}_{ij}, & \lceil x_j \rceil > 0. \end{cases}$$

Observe that these matrices are defined in terms of the solutions they lead to. Our solution technique bypasses the search for the optimal settings of G and E and yields the optimal solution directly. By symmetry, one can see that a similar result holds for d^{opt}.

6.7 Explanation and Example of the Solution Method

Theorem 6.5 describes an efficient technique for computing the target state of an arbitrary switch-level network. First we compute the vector r by applying the relaxation method to the equation

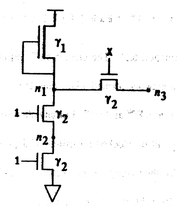
$$r = E^{min} \bullet \| x \| \uparrow \| y \| \uparrow G^{min} \bullet r, \tag{6.33}$$

which gives the same result as equation 6.31. The elements of r equal the strengths of the steady state signals in the logical conductance network formed when only transistors in the 1 state are conducting, since the equation involves the matrices E^{min} and G^{min}. For any allowable values of G and E, any signal on node n_i with strength less than r_i will be blocked regardless of the conductances of transistors in the X state and hence can be killed. We then compute the vectors u^{opt} and d^{opt} by applying the conditioned relaxation method to equations 6.29 and 6.30. These computations consider transistors in both the 1 and

X state to form logical conductances equal to their strengths, giving G^{max} and E^{max} in the first argument to *block* in both equations, but any strength value on node n_i less than r_i is blocked. The computations are performed separately for the 1 and 0 signals, so that a signal (represented by a strength value) will not be killed if it could be the dominant signal for some set of transistor conductances. Signals of state X can be considered to have both state 1 and state 0, and hence they enter into both computations. Once these strength values have been found, they can be combined to give the target states.

This technique requires no enumeration over possible sets of transistor conductances whatsoever. This method cannot be formulated as a more intuitive method in the signal algebra, such as the method shown in Corollary 6.4.1, because our optimization technique does not correspond to the operation of any single logical conductance network. Furthermore, no simpler method has been obtained for restricted networks, because transistors in the X state in some ways resemble weak transistors. That is, the signal paths in a restricted network cannot be analyzed independently when transistors in the X state are present.

Fig. 6.3. Switch-Level Network Example



This method can be illustrated by the example shown in Figure 6.3, representing an nMOS Nand gate with both inputs equal to 1 connected to a pass transistor in state X. Assume that node n_3 has size κ_1 and initial state 0. The recurrence equation for r can be written as:

$$\begin{aligned} r_1 &= \gamma_1 \uparrow (\gamma_2 \downarrow r_2) \\ r_2 &= \gamma_2 \uparrow (\gamma_2 \downarrow r_1) \\ r_3 &= \kappa_1. \end{aligned}$$

The minimum solution of this set of equations is $r_1 = r_2 = \gamma_2$, and $r_3 = \kappa_1$. The recurrence equation for u^{ext} can be written as

$$\begin{aligned} u_1 &= block(\gamma_1 \uparrow (\gamma_2 \downarrow u_2) \uparrow (\gamma_2 \downarrow u_3), \gamma_2) \\ u_2 &= block(\gamma_2 \downarrow u_1, \gamma_2) \\ u_3 &= block(\gamma_2 \downarrow u_2, \kappa_1). \end{aligned}$$

The minimum solution of this set of equations is $u^{ext}_1 = u^{ext}_2 = u^{ext}_3 = 0$, indicating that regardless of the conductance formed by the pass transistor, no signal with state 1 or X can form on any nodes. Even though the pullup transistor provides a signal of strength $+\gamma_1$, our computation correctly recognizes that this signal will be blocked by the signal $-\gamma_2$. The recurrence equation for d^{ext} is

$$\begin{aligned} d_1 &= block((\gamma_2 \downarrow d_2) \uparrow (\gamma_2 \downarrow d_3), \gamma_2) \\ d_2 &= block(\gamma_2 \uparrow (\gamma_2 \downarrow d_1), \gamma_2) \\ d_3 &= block(\kappa_1 \uparrow (\gamma_2 \downarrow d_2), \kappa_1). \end{aligned}$$

The minimum solution of this set of equations is $d^{ext}_1 = d^{ext}_2 = d^{ext}_3 = \gamma_2$. Thus, since these values are all nonzero, while the values of u^{ext} are all 0, all three nodes have a target state 0.

If the same network has initial state 1 for n_3, we would find that $u^{ext}_3 = \kappa_1$, while all other elements of u^{ext} and d^{ext} have the same values as before. This gives target states $\tilde{y}_1 = \tilde{y}_2 = 0$, and $\tilde{y}_3 = X$, indicating that the unknown conductance of the pass transistor creates an ambiguity in the target state of node n_3.

6.8 Properties of the Target State

Now that we have a mathematical description of the target state, several useful properties can be demonstrated.

6.8.1 Monotonicity

Our partial ordering of signal states ranks states according to how well defined they are. The state $\perp$ is underfined, i.e. it represents an absence of information. This state will never appear on a node in a switch-level network, because all nodes store information dynamically and hence can never be devoid of state. The states 0 and 1 are well defined, i.e. they represent a consistent degree of information. The state X is overdefined, i.e. it represents conflicting information. The following theorem shows that the target state function is monotonic for this ordering. This indicates that setting some node or transistor to X can only lead to target states for some nodes equal to X which would otherwise equal Boolean values.

Theorem 6.6.
If $x \leq x'$, $y \leq y'$, $z \leq z'$ then

$$target(x, y, z) \leq target(x', y', z').$$

Proof of Theorem 6.6:

This theorem can be proved by comparing the derivation of the target state for initial values x, y, z (which will be shown with unprimed values), with the derivation for initial values x', y', z' (which will be shown with primed values.) Compare the function

$$g_1(u) = block(E^{max} \bullet \lceil x \rceil \uparrow \lceil y \rceil \uparrow G^{max} \bullet u, r)$$

with the function

$$g_1'(u) = block(E^{max'} \bullet \lceil x' \rceil \uparrow \lceil y' \rceil \uparrow G^{max'} \bullet u, r'),$$

where

$$r \;=\; C^{out*}\cdot(E^{out}\cdot\|x\| \uparrow \|y\|),$$

and

$$r' \;=\; C^{out'*}\cdot(E^{out'}\cdot\|x\| \uparrow \|y\|),$$

One can see from equations 6.1, 6.2, 6.3, and 6.4, that if $x \le x'$ then

$$E^{out} \ge E^{out'}$$
$$C^{out} \ge C^{out'}$$
$$E^{max} \le E^{max'}$$
$$C^{max} \le C^{max'}.$$

Since the strengths of the initial stimulus signals are determined only by the node types and sizes, $\|x\| = \|x'\|$, and $\|y\| = \|y'\|$. Therefore $r \ge r'$. Furthermore, if $x \le x'$, then $x \le x'$ and therefore $\lceil x \rceil \le \lceil x' \rceil$. Similarly, $\lceil y \rceil \le \lceil y' \rceil$. Therefore $g_1 \le g_1'$ and by Theorem 6.3, $u^{out} \le u^{out'}$. By similar reasoning, one can see that $g_0 \le g_0'$ and therefore $d^{out} \le d^{out'}$. Observe that the function of b whose value is $\blacktriangleleft + b \blacktriangleright$ is monotonic, and therefore $\blacktriangleleft + u^{out} \blacktriangleright \le \blacktriangleleft + u^{out'} \blacktriangleright$ and by similar reasoning $\blacktriangleleft - d^{out} \blacktriangleright \le \blacktriangleleft - d^{out'} \blacktriangleright$. Finally, by the monotonicity of $\sqcup$

$$target\,(x, y, z) \;=\; \blacktriangleleft + u^{out} \blacktriangleright \sqcup \blacktriangleleft - d^{out} \blacktriangleright \;\le\; \blacktriangleleft + u^{out'} \blacktriangleright \sqcup \blacktriangleleft - d^{out'} \blacktriangleright \;=\; target\,(x', y', z'). \qquad \blacksquare$$

The monotonicity property then extends to the functions $step_x$ and $phase$.

Corollary 6.6.1.
If $x \le x'$ and $y \le y'$ then

$$step_x(y) \;\le\; step_{x'}(y').$$

Proof of Corollary 6.6.1:

One can see from the table in Section 2.5 that $trans(x, y) \leq trans(x', y')$. Therefore

$$step_x(y) = target(x, y, trans(x, y)) \leq target(x', y', trans(x', y')) = step_{x'}(y'). \quad \blacksquare$$

Corollary 6.6.1.
If $x \leq x'$ and $y \leq y'$ then

$$phase(x, y) \leq phase(x', y').$$

Proof of Corollary 6.6.1:

By Corollary 6.6.1 and induction on k,

$$step_x^k(y) \leq step_{x'}^k(y').$$

Therefore

$$phase(x, y) = \lim_{k \to \infty} step_x^k(y) \leq \lim_{k \to \infty} step_{x'}^k(y') = phase(x', y'). \quad \blacksquare$$

These results show that the presence of an X value on a node can only lead to new network states which have some nodes set to X which would otherwise be set to Boolean values.

6.8.2 Stability of the Target State

The target state is claimed to be the set of states which the normal nodes would eventually reach if the input nodes and transistor states were held in states x and z, and the normal nodes were initialized to state y. To really prove this, we must show that once the network reaches the target state, it will stay there until some input node or transistor changes state. While this stability can readily be seen for the steady state of a logical conductance network, it is less clear for the target state of a switch-level network. The following theorem eliminates any such doubts.

Theorem 6.7.
If $\tilde{y} = target(x, y, z)$, then $\tilde{y} = target(x, \tilde{y}, z)$.

The proof of this theorem is also given in Appendix II. It involves showing that the terms $\lceil y \rceil$ and $\lfloor y \rfloor$ in equations 6.29 and 6.30 can be replaced with terms $\lceil \tilde{y} \rceil$ and $\lfloor \tilde{y} \rfloor$ where $\tilde{y}$ is the vector of signals with ith element having state equal to the target state $\tilde{y}_i$ and strength equal to the node size cap_i.

6.9 Summary

Our entire development of the switch-level model so far can be summarized by three equations

$$r = E^{min} \cdot \| x \| \uparrow \| y \| \uparrow G^{min} \cdot r \tag{6.33}$$

$$u = block(E^{max} \cdot \lceil x \rceil \uparrow \lceil y \rceil \uparrow G^{max} \cdot u, r) \tag{6.29}$$

$$d = block(E^{max} \cdot \lfloor x \rfloor \uparrow \lfloor y \rfloor \uparrow G^{max} \cdot d, r). \tag{6.30}$$

By finding the minimum solution of the first equation and then using this value in computing the minimum solutions of the other two, we obtain two vectors of strength values u^{opt} and d^{opt} from which the target state for each node can be computed as

$$\tilde{y}_i = \begin{cases} 1, & u^{opt}_i > 0 \text{ and } d^{opt}_i = 0 \\ 0, & d^{opt}_i > 0 \text{ and } u^{opt}_i = 0 \\ X, & \text{else.} \end{cases} \tag{6.26}$$

Consider how far we have progressed from the electrical circuit-oriented view of the switch-level model provided by the original definition of the target state in Chapter 2. This new method involves only simple operations in a discrete algebra, and the equations can be solved by a straightforward iterative method. Furthermore, it finds whether nodes are sensitive to the unknown conductances formed by transistors in the X state (and hence should have target state X) without enumerating over possible combinations of transistor conductances. Thus, it can be implemented by a very efficient computer algorithm as is shown in the next chapter.

7. Simulation Algorithms

7.1 Introduction

Theorem 6.5 defines a straightforward method for computing the target state function, and this method can be implemented by an efficient algorithm to serve as the basis of a switch-level simulator. By exploiting the locality of both the interconnections and the activities in the network, the program can achieve a performance comparable to logic gate simulators. First a unit delay simulation algorithm is presented which provides the same functionality as the program MOSSIM [9] for designs which can be described in the MOSSIM network model. Next, it is shown that a slight modification yields a simulator with a timing model similar to Terman's program [5], although the functionality of the two algorithms differ significantly. The new algorithm differs greatly in its style from both of these previous algorithms, largely because it is based on solving equations in a well-defined mathematical domain rather than on the intuitive ideas of the simulator designers. These simulation algorithms are compared and contrasted toward the end of the chapter. Some performance data from MOSSIM are presented to demonstrate the performance characteristics of switch-level simulation and how it compares to logic gate simulation. All algorithms are presented as "Pidgin Algol" programs as defined in Aho, Hopcroft, and Ullman [1].

Before delving into the details of the simulation algorithm, let us consider its intended mode of use. Suppose the design to be simulated will operate as a synchronous circuit with a conservative clocking scheme. That is, some external set of clock signals will be provided through input nodes which control the sequential operation of the circuit such that as long as these clocks run slowly enough, no timing errors can occur. Each clock cycle can be subdivided into a set of *simulation phases* (called "epochs" in Mead and Conway [37]) where during each phase, all clock and data inputs remain constant. For example, a two-phase, nonoverlapping clock contains four such simulation phases:

	Phase			
	1	2	3	4
Phi1	0	1	0	0
Phi2	0	0	0	1

During each phase, the circuit has sufficient time to stabilize.

To model the functionality of such a circuit, a simulator can simply compute the state in which the network would settle for each phase of each clock cycle, setting the clock and data inputs to new values between the phases. The function *phase*, defined in Chapter 2 as $phase(x, y) = \lim_{k \to \infty} step_x(y)$ serves this purpose. To the user, this technique provides the effect of a unit delay timing model in which transistors switch one time unit (i.e. on the next computation of $step_x$) after their gate nodes change state. Such a technique provides only limited information about the speed of the actual circuit, but gives an indication of the function computed. The characteristics of this and other timing models are discussed in Chapter 8.

This technique has been applied to simulating self-timed systems [36] as well, in which activities may occur independently and asynchronously. Each phase then corresponds to a particular setting of the input data and control signals, and it is assumed that the circuit will settle before the inputs are changed. Although actual circuits may not obey these assumptions, almost all can be modeled as if they did.

7.2 Complexity Model

In a switch-level network, each node could be connected to every other node by any number of transistors, giving an unbounded number of transistors relative to the number of nodes. In practice, however, the number of transistors grows only linearly with the number of nodes due to the limited connectivity allowed by a two-dimensional integrated circuit chip and to electrical and functional considerations. To evaluate simulation algorithms, we should have a model of the complexity of networks which more closely matches actual circuits. The following set of assumptions has been observed to hold for a variety of designs, although it has not been subjected to a rigorous study.

Define the *connectivity set* of a node as the set of transistors for which the node serves as the source or drain connection and the *connectivity degree* as the size of this set. The *fanout set* of a node is defined as the set of transistors for which the node serves as the gate connection, and the *fanout degree* is defined as the size of this set. An informal study of a variety of designs has shown that almost all normal nodes have connectivity degree less than 5. Exceptions include large busses and the output nodes of large Nor gates.[1] Input nodes, especially VDD and GND, however, may have a high degree of connectivity. A similar statistic holds for fanout degree with the exception of nodes providing major control signals such as clocks and reset or enabling commands.

We will assume the network may contain O(1) (i.e. a constant number) of input nodes each with O(n) fanout and connectivity degree, where n is the number of normal nodes. An O(1) subset of the normal nodes may also each have O(n) fanout and connectivity degree, but the remaining normal nodes must each have O(1) fanout and connectivity degree. From either the fanout or the connectivity assumptions, one can see that the network can contain only O(n) transistors.

The sparseness of interconnections in a logic design leads to a localization of the activities. When a node changes state, generally only a small number of nodes will be directly affected. Furthermore, in most synchronous designs, each logic element will be activated only a small number of times during each clock cycle. That is, during a single simulation phase, information will only propagate from the outputs of one set of storage elements through some combinational logic to the inputs of other (or perhaps the same) storage elements. Even allowing for a small number of dynamic hazards (transient pulses caused by unequal path delays), each node will change state only O(1) times during each phase. In fact, experience has shown that often significantly fewer state changes occur. For example, in a random access

---

1. Structures involving many transistors of the same strength and type connected in parallel, such as large Nor gates could be simulated more efficiently if they were modeled by special "multi-transistors" which have multiple gate nodes, any of which can activate the switch. A simple count of the number of gate nodes in the X and 1 state would indicate the state of such an element.

memory, only a small percentage of the nodes change state during each clock cycle. While this example represents an extreme case, most networks contain only a small number of active elements at any given time.

7.3 Sparse and Incremental Equations

Our complexity model shows that the connectivity in the network is very sparse, and that changes in the network state occur only *incrementally*, i.e. a small number of nodes at a time. A well-designed simulation algorithm can exploit both of these reductions in complexity and thereby achieve considerably better performance than would a naive implementation of the matrix equations. These techniques will be demonstrated by developing algorithms for solving sparse and incremental equations in the strength algebra.

7.3.1 Sparse Equations

Suppose we wish to find the minimum solution of the equation $\mathbf{a} = f(\mathbf{a})$ where $f:\mathfrak{I}^n \rightarrow \mathfrak{I}^n$ can be expressed as

$$[f(\mathbf{a})]_i = b_i \uparrow \uparrow_{n_j \in P_i} f_{ij}(a_j) \tag{7.1}$$

The set P_i is called the *adjacency* set of node n_i and in our application will equal the set of normal nodes connected to the node by transistors in the 1 or X state. We will assume that all connections are bidirectional, i.e. $n_i \in P_j$ if and only if $n_j \in P_i$. Furthermore the function f is assumed to be both *monotonic* and *passive*. The general definition of passive functions is described in Appendix II, but for the function f above implies that for all indices i and j and all strength values s, $f_{ij}(s) \leq s$.

As an example the function f_1 defined in equation 6.20 as

$$f_1(a) = block(\lceil b \rceil \uparrow G \bullet a, r) \tag{6.20}$$

can be expressed in this form with $b_i = block(\lceil b_i \rceil, r_i)$ and $f_{ij}(a_j) = block(g_{ij} \downarrow a_j, r_i)$. One can see that f_1 is both monotonic and passive. This example also motivates the term "passive", which corresponds to the property that a signal coupled through a logical conductance can never be increased in strength. This example also demonstrates how the locality of interconnections in the network lead to sparse matrix equations. Element i,j of matrix G can only be greater than 0 if a transistor in the 1 or X state connects nodes n_i and n_j. By our assumptions about network connectivity, $|P_i|$ can be O(n) for only O(1) values of i, and must be O(1) otherwise. Therefore $\Sigma|P_i|$ must be O(n).

The following program solves this equation where S denotes some data structure such as a stack in which elements can be inserted (push) and removed (pop) in unit time. The order in which elements are removed is unimportant.

```
procedure SOLVE1(f):
begin
   S ← Ø;
   for i ← 1 until n do
      begin
         a_i ← b_i;
         push(S, n_i);
         done_i ← false
      end;
   while S ≠ Ø do
      begin
         n_j ← pop(S);
         if done_j = false then
            begin
               done_j ← true;
               for each n_i ∈ P_j do
                  if f_ij(a_j) > a_i then
                     begin
                        a_i ← f_ij(a_j);
                        done_i ← false;
                        push(S, n_i)
                     end
            end
      end;
   return(a)
end
```

The procedure SOLVE1 resembles the relaxation method outlined in Chapter 4, except that it tries to minimize the amount of computation by computing the effects of a node value on adjacent nodes only when the value changes. It starts by setting all nodes to the initial values given by **b** and placing the nodes in the list S. At any time, any node n_j in the list for which $done_j$ equals *false* has had a new value assigned to a_j, the effect of which has not been propagated to neighboring nodes. The procedure performs a series of relaxations, each of which starts by selecting a node n_j from S such that $done_j$ equals *false*. The effect of the value a_j on each neighboring node in P_j, i.e. $f_{ij}(a_j)$ is computed, and if this exceeds the previous value on the neighboring node, the neighbor is updated and placed in the list with $done_i$ set to *false*. This may lead to duplications in the list S, because n_i may already be on it. The flag $done_i$, however, provides a means of checking whether the effects of the node value on adjacent node values have already been

computed. This technique eliminates the need to test a node for membership in S before adding it to S.[1] These relaxations continue until all nodes are consistent with one another, i.e. any further relaxations would have no effect.

The correctness of the procedure SOLVE1 relies only on the monotonicity of f. This can be proved by inductive assertion on the number of relaxation steps (i.e. executions of the while loop body.) Let $\mathbf{a}^k$ equal the value of the array $\mathbf{a}$ after k steps. It is claimed that for any k, $\mathbf{b} \leq \mathbf{a}^k \leq \mathbf{a}^{min}$ and for any j such that $done_j$ equals *true*, $f_{ij}(a_j) \leq a_i$ for all i in P_j. This clearly holds at the start, because $\mathbf{b} = \mathbf{a}^0 = f(0) \leq \mathbf{a}^{min}$ and $done_j$ equals *false* for all j. Now suppose this assertion holds after relaxation step k. A relaxation step involves propagating values according to the set of functions f_{ij}, and therefore $\mathbf{a}^k \leq \mathbf{a}^{k+1} \leq \mathbf{a}^k \uparrow f(\mathbf{a}^k)$, which gives

$$\mathbf{b} \leq \mathbf{a}^k \leq \mathbf{a}^{k+1} \leq \mathbf{a}^k \uparrow f(\mathbf{a}^k) \leq \mathbf{a}^{min} \uparrow f(\mathbf{a}^{min}) = \mathbf{a}^{min}.$$

Furthermore, the second part of the assertion clearly holds, because the program sets $done_j$ to *false* any time a_j has been changed and only sets it to *true* once the effects of the new value have been computed. By induction the assertion must hold when the procedure terminates with $done_j$ equal to *true* for all j. This implies that for the final value of the vector $\mathbf{a}$, $a_i \geq f_{ij}(a_j)$ for all i and j. We also know that $a_i \geq b_i$ for all i, and hence $\mathbf{a} \geq f(\mathbf{a})$. By the monotonicity of f and induction on k, $\mathbf{a} \geq f^k(\mathbf{a})$ for all k, and therefore

$$\mathbf{a} \geq \lim_{k \to \infty} f^k(\mathbf{a}) \geq \lim_{k \to \infty} f^k(0) = \mathbf{a}^{min}.$$

Combining this inequality with the inequality of the induction assertion gives $\mathbf{a} = \mathbf{a}^{min}$ when SOLVE1 terminates.

---

1. One could test the flag $done_i$ before inserting a node in S to avoid duplication in S, but the method given here leads more naturally to our further developments.

To analyze the complexity of this procedure, observe that a node is pushed on S only when its value is increased. Thus each node v_i can only be pushed a maximum of $|P_i|$ times; each time causing at most one relaxation step requiring $|P_i|$ operations. Therefore the algorithm has complexity less than or equal to $O(\Sigma_i |P_i|^2) = O(|T|\Sigma_i |P_i|) = O(|T|n)$.

The following example shows that this degree of complexity can actually be realized. Define $\mathbf{h}$ as a vector of decreasing strength values followed by $\mathbf{0}$'s, i.e.

$$h_i = \begin{cases} \gamma_{p+1-i} & 1 \le i \le p \\ \gamma_{p+q+1-i} & p+1 \le i \le p+q \\ 0 & p+q+1 \le i \le n. \end{cases}$$

Let $P_1 = \{v_2\}$, $P_n = \{v_{n-1}\}$, and $P_i = \{v_{i-1}, v_{i+1}\}$ for $1 < i < n$. Define f_B as the identity function for $v_i \in P_i$. This example corresponds to a linear chain of normal nodes with the nodes having initial signals of decreasing strength. Suppose that S is implemented as a stack and that nodes are selected from each set P_i in order of their subscripts. SOLVE1 will first set all nodes v_i such that $i \ge p+q$ to v_1, and then it will set all nodes v_i such that $i \ge p+q-1$ to v_2, and so on through all possible strength values until finally all nodes are set to γ_p. Thus, the worst case complexity of a SOLVE1 equals $O(|T|n)$.

For most MOS networks we can assume that T is a very small set. For example, the network model of MOSSIM can be implemented with $T = \{0, \gamma_1, \gamma_2\}$. Therefore SOLVE1 provides a linear and hence optimal solution. Nonetheless, our worst case example shows that SOLVE1 can waste much effort in propagating values which will only be overridden later. A slight refinement, however, leads to an algorithm with complexity $O(n)$. It replaces the list S with and array of lists B, with one list for each possible strength value.

```
procedure SOLVE2(f):
begin
   for each s ∈ 𝒮 do B[s] ← ∅;
   for i ← 1 until n do
      begin
         a_i ← b_i;
         push(B[a_i], n_i);
         done_i ← false
      end;
   PROPAGATE(B, a, f);
   return(a)
end
```

In this program node n_i is inserted into the stack corresponding to the initial value b_i. The procedure PROPAGATE, defined below, then spreads these values through the network.

```
procedure PROPAGATE(B, a, f):
begin
   s ← γ_p;
   repeat
      begin
         while B[s] ≠ ∅ do
            begin
               n_j ← pop(B[s]);
               if done_j = false then
                  begin
                     done_j ← true;
                     [init_j ← false;]
                     for each n_k ∈ P_j do
                        if f_jk(a_j) > a_k then
                           begin
                              a_k ← f_jk(a_j);
                              push(B[a_k], n_k)
                           end
                  end
            end;
         s ← pred(s)
      end
   until s = 0
end
```

The function *pred* in this procedure is the predecessor function for $\mathcal{S}$. That is, *pred* (0) = 0, and for s > 0, *pred* (s) equals the greatest element of $\mathcal{S}$ less than s. This function is used to enumerate the strength values in descending order. The line enclosed in square brackets is required for our next extension of the

algorithm.

The procedure SOLVE2 operates much like SOLVE1, except that a node is selected for relaxation only if it has maximal strength of all nodes for which $done_i$ equals *false*. It does this by keeping the nodes on separate lists according to their strengths and going through these lists in decreasing order. It relies on the passiveness of f to assure that if a node of maximal strength is selected for relaxation, no nodes will be set to greater strength during the remainder of the computation. Therefore each node is selected for relaxation only once, and hence the complexity of SOLVE2 equals $O(|P_i|) = O(n)$. It is unclear whether SOLVE2 will actually achieve a better performance than SOLVE1, because the cost of implementing the array of stacks might exceed the gain in efficiency. This depends on the details of the programming language as well as on the networks to be simulated. Nonetheless, we shall pursue the algorithm for SOLVE2 for further development.

7.3.2 Incremental Equations

As an extension of this technique, suppose that we have computed the minimum solution $\mathbf{a}^{min}$ of the equation $\mathbf{a} = f(\mathbf{a})$ with f defined by equation 7.1 and now wish to find the minimum solution $\mathbf{a}^{min'}$ of the equation $\mathbf{a} = f'(\mathbf{a})$, where

$$[f'(\mathbf{a})]_i = b'_i \uparrow \uparrow_{n_j \in P'_i} f'_{ij}(a_j),$$

and f' obeys the same restrictions as f. Furthermore, assume that b'_i differs from b_i for only a small number of values of i, that f'_{ij} differs from f_{ij} for only a small number of values of i and j, and that $P_i \neq P'_i$ for only a small number of values of i.

For the example of the function f_1 given earlier, the differences between b_i and b'_i reflect changing input node states or changing connections to input nodes. The differences between f_{ij} and f'_{ij} and between P_i and P'_i reflect changing connections between normal nodes.

The differences between the functions f and f' can be regarded as perturbations of the function f. That is node n_i is perturbed if $b_i \neq b'_i$, $P_i \neq P'_i$, $f_{ij} \neq f'_{ij}$, or $f_{ji} \neq f'_{ji}$. Such a perturbation can only affect nodes in the *vicinity* of node n_j, where n_j is defined to be in the vicinity of n_i if there exists some path from n_j to n_i in the graph with edges defined by the connectivity sets P'_k. Thus, the new equation can be solved "incrementally", by only computing new values for those nodes in the vicinity of a perturbed node.

```
procedure INCR_SOLVE(f, f', a):
begin
    E ← Ø;
    for each i such that b_i ≠ b'_i or P_i ≠ P'_i or f_ij ≠ f'_ij or f_ji ≠ f'_ji do
        begin
            E ← E ∪ { n_i };
            done_i ← false
        end;

    for each s ∈ 𝒮 do B[s] ← Ø;
    for each n_i ∈ E such that done_i = false do
        begin
            INITIALIZE(n_i, b', a, B);
            PROPAGATE(B, a, f')
        end;
        return(a)
end
```

The procedure INITIALIZE sets all nodes in the vicinity of n_i to the initial values given by b', using a depth-first search technique for finding all connected nodes in a graph as is described in Aho, Hopcroft, and Ullman [1]. The flag $init_i$ is used to indicate whether the node has already been initialized. It is assumed to equal *false* at the start of the program and is also set to *false* by the procedure PROPAGATE once the relaxations begin. The procedure also places each initialized node into the appropriate list in the array B.

```
procedure INITIALIZE(n_i, b', a, B):
    if init_i = false
        then
            begin
                init_i ← true;
                a_i ← b'_i;
                push(B[a_i], n_i);
                for each n_j ∈ P_i do
                    INITIALIZE(n_j, b', a, B)
            end
```

The procedure PROPAGATE has already been given. It will take the initial values set by INITIALIZE and spread them through the network in the vicinity of the perturbation. Observe that the flag $done_i$ is used for two purposes. It is used by the procedure INCR_SOLVE to avoid redundant computations when two perturbed nodes are in the same vicinity. It is also used by the procedure PROPAGATE to take care of the case where a node has been moved to a new list but has not been deleted from the old.

The complexity of the procedure INCR_SOLVE is proportional to the number of nodes in the vicinity of a perturbed node. This can range from O(1) to O(n). In any case, the procedure is close to optimal, because it only looks at nodes in the vicinity of perturbations. A truly optimal algorithm would only look at a node if its value will change, but this is hard to achieve.

7.4 Unit Delay Simulation Algorithm

Theorem 6.5 shows that the target state can be computed by solving equations for $\mathbf{r}$, $\mathbf{u}^{\infty}$, and $\mathbf{d}^{\infty}$ in the strength algebra. We will now drop the superscripts on $\mathbf{u}$ and $\mathbf{d}$. As the phase simulation progresses, these values change only incrementally. Thus the techniques developed in the previous section lead directly to an algorithm for a switch-level simulator. In the following programs it is assumed that the network structure and state information as well as the vectors $\mathbf{r}$, $\mathbf{u}$ and $\mathbf{d}$ are available as global variables and need not be passed as arguments. Only an outline of the algorithm will be given here, because it involves many details and requires further development of the proper set of data structures.

The program PHASE shown below computes the function

$$phase(x, y) = \lim_{k \to \infty} step_x^k(y). \qquad (2.3)$$

It takes advantage of the fact that each computation of $step_x$ involves only incremental changes to the network state. The procedure is given a list of node-state pairs as an argument. Each element of this list is of the form $\langle i, x \rangle$, indicating a new setting for an input node, or $\langle n_j, y \rangle$, indicating a new setting for a normal node. The variable "newval" represents some element of this list. In general, only input nodes will be changed, because in an actual circuit only these nodes are accessible externally.

```
procedure PHASE(A):
begin
    E ← Ø;
    for each newval ∈ A do
        begin
            SET_NODE(newval);
            E ← E ∪ PERTURB_NODE(newval);
            SET_TRANSISTORS(newval);
            E ← E ∪ PERTURB_TRANSISTORS(newval)
        end;

    while E ≠ Ø do
        E ← STEP(E)
end
```

The procedure SET_NODE updates the node state, and the procedure PERTURB_NODE places those normal nodes perturbed by this change in the list E as well as sets the flags $done_i$ to *false* for these nodes. That is, a changing input node will perturb all normal nodes connected by conducting transistors, while a changing normal node will perturb only itself. The procedure SET_TRANSISTORS updates the state of every transistor in the fanout set of the node, while the procedure PERTURB_TRANSISTORS places each normal node for which a transistor in its connectivity set has changed state in the list E and sets the flags $done_i$ to *false* for these nodes. The procedure STEP simulates the effects of the perturbations on the nodes in E which then creates a new set of perturbations to be simulated. This process continues until a stable state is reached, i.e. no perturbations remain.

The procedure STEP is shown below.

```
procedure STEP(E):
begin
    A ← ∅;
    for each n_i ∈ E such that done_i = false do
        A ← A ∪ UPDATE(n_i);

    E ← ∅;
    for each newval ∈ A do
        begin
            SET_TRANSISTORS(newval);
            E ← E ∪ PERTURB_TRANSISTORS(newval)
        end;
    return(E)
end
```

The procedure STEP selects a node from the list of perturbed nodes and calls the procedure UPDATE to compute the new states of all nodes in the vicinity of the selected node. Those nodes which change state during this process are accumulated in a list A. Once the effects of all perturbations have been simulated, the transistors in the fanout sets of nodes in A are set to their new states. This will cause new perturbations which are accumulated in the new list E. Observe that the procedure PERTURB_NODE need not be called, because by Theorem 6.7. the target state will remain stable unless either an input node or transistor changes state. By changing the transistor states only after all perturbations have been simulated, the procedure STEP creates the effect of all node states changing simultaneously and then all transistor states changing simultaneously. This implements a timing model in which transistors switch one time unit after their gate nodes change state.

The procedure UPDATE is shown below. It applies the technique shown in the procedure INCR_SOLVE to solve the equations

$$\mathbf{r} = \mathbf{E}^{min} \bullet \| x \| \uparrow \| y \| \uparrow \mathbf{G}^{min} \bullet \mathbf{r} \tag{6.33}$$

$$\mathbf{u} = block\,(\mathbf{E}^{max} \bullet \lceil x \rceil \uparrow \lceil y \rceil \uparrow \mathbf{G}^{max} \bullet \mathbf{u}, \mathbf{r}) \tag{6.29}$$

$$\mathbf{d} = block\,(\mathbf{E}^{max} \bullet \lfloor x \rfloor \uparrow \lfloor y \rfloor \uparrow \mathbf{G}^{max} \bullet \mathbf{d}, \mathbf{r}). \tag{6.30}$$

From these the value of the target state is computed for each node as

$$\tilde{y}_i = \begin{cases} 1, & u_i > 0 \text{ and } d_i = 0 \\ 0, & d_i > 0 \text{ and } u_i = 0 \\ X, & \text{else.} \end{cases} \quad (6.26)$$

Each node which changes state is placed in a list, along with its new value.

```
procedure UPDATE(n_i)
begin
    for each s ∈ B[s] do B[s] ← ∅;
    INITIALIZE_R(n_i, B);
    PROPAGATE_R(B);

    INITIALIZE_U(n_i, B);
    PROPAGATE_U(B);

    INITIALIZE_D(n_i, B);
    PROPAGATE_D(B);

    A ← UPDATE_STATE(n_i);

    return(A)
end
```

The remaining procedures will not be given.

The speed of the procedure UPDATE could be improved for the case where no paths of conducting transistors from n_i to input nodes or other normal nodes contain transistors in the X state. Suppose furthermore that all paths from n_i to other normal nodes contain only transistors of strength γ_p. Then Corollary 6.1.1 shows that the steady state signal for every node connected by some path to n_i can be found by combining the initial signals on these nodes using the operation $\vee$, and the state of this signal equals the target state of all of these nodes. The procedure INITIALIZE_R could check whether this particular condition exists, and if so a procedure could be called to perform this computation, and then the nodes could be set to the state of this signal. This computation involves considerably less effort than computing r, u, and d. Considering that in most simulations the X state arises only rarely once the X's present at the start of the simulation have been forced away, such an optimization could provide a substantial speed gain. For the case where some normal nodes are connected by transistors with strength

less than γ_p (i.e. an unrestricted network) we could employ the method of Corollary 6.4.1 to find the steady state signals and consequently the target states. This method, however, may not provide a significant savings over the original method.

Our network complexity gives a rather weak bound on the complexity of PHASE. If we assume that each node changes state only O(1) times, then the total number of perturbations to the network will be proportional to the sum of the fanout degrees of all nodes, which is O(n). However, a perturbation may require O(n) operation to simulate its effects, although such cases are rare. This gives a total complexity of $O(n^2)$. Such complexity is achieved only by highly contrived examples, however. Experience has shown that typical networks require at most O(n) operations per phase.

7.5 Pseudo Unit Delay Simulation Algorithm

The algorithm presented in the previous section carefully holds all transistors fixed until all perturbations have been simulated, thereby creating the effect of all nodes changing state simultaneously and then all transistors switching simultaneously one time unit later. If we instead switch transistors immediately after their gate nodes change state, we obtain an algorithm with many characteristics of a unit delay simulator but in which all events are ordered. The characteristics of this timing model are discussed further in Chapter 8.

```
procedure PHASE2(A):
begin
    E ← ∅:
    for each newval ∈ A do
    begin
        SET_NODE(newval):
        E ← E ∪ PERTURB_NODE(newval);
        SET_TRANSISTORS(newval);
        E ← E ∪ PERTURB_TRANSISTORS(newval)
    end;
    while E ≠ ∅ do
    begin
        n_1 ← dequeue(E):
        if done_1 = false then
        begin
            A ← UPDATE(n_1):
            for each newval ∈ A do
            begin
                SET_TRANSISTORS(newval);
                E ← E ∪ PERTURB_TRANSISTORS(newval)
            end
        end
    end
```

In this procedure, elements are removed from the list E in first-in, first-out order so that the effects of simultaneous perturbations will be simulated before any subsequent effects are simulated. As a result, the algorithm provides a similar timing model to the unit delay algorithm, even though the list E evolves continuously during the simulation phase rather than being repeatedly filled and emptied.

7.6 Comparison to Other Switch-Level Simulators

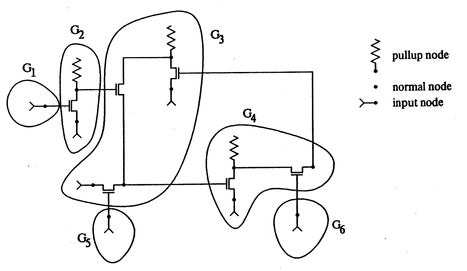
Both MOSSIM [9] and the simulator developed by Terman [5] are designed along similar lines to the ones described here. A comparison between these three simulators serves to highlight some issues in simulator design.

7.6.1 MOSSIM

For networks which can be described in the MOSSIM network model, the unit delay simulation algorithm presented here will produce the same results as MOSSIM. The two programs, however, differ greatly in their internal structure.

MOSSIM precedes the simulation by a relatively complex network analysis, primarily to partition the network in a way which allows selective updating. At the start of this analysis each input node is replicated to give a separate copy for each transistor in its connectivity set. Then the network is partitioned into *transistor groups* with each group corresponding to a connected component in the undirected graph with a vertex for each node and an edge between each pair of vertices corresponding to the source and drain nodes of a transistor. This partitioning divides the network into components which interact only through fanout connections, i.e. from a node in one group to a transistor gate in another.

Fig. 7.1. A MOSSIM Network Partitioned into Transistor Groups



Since we assume the gate node is electrically isolated from the source and drain, such a connection is purely unidirectional and depends only on the state of the node signal and not on its strength. Thus each transistor group can be viewed as a logic block with inputs, outputs, and internal state and which communicates with other blocks only with logic values 0, 1, and X. An example of a MOSSIM network partitioned into transistor groups is shown in Figure 7.1. A similar partitioning method has been used by researchers at the Nippon Electric Company in an analog simulator to achieve a localization of activities [42]. Although our new algorithm does not require this partitioning, such a technique provides a way of introducing some degree of structure into the network. This structuring has several applications that will be described shortly.

During the simulation a transistor group need only be simulated when one of its inputs has changed. This tends to restrict the simulation to the active portions of the network, thereby achieving some of the effect of the incremental updating technique used in the new algorithm. MOSSIM, however, can only take advantage of the *static* locality in the network, i.e. that which can be detected without considering transistor states. The new algorithm also takes advantage of *dynamic* locality in which the source and drain nodes of a transistor in the 0 state are also considered electrically isolated. Thus, only nodes connected by paths of conducting transistors to a perturbed node need be updated. This added selectively should yield some gain in speed.

MOSSIM also uses a significantly different method of updating a set of nodes following a perturbation. It exploits the fact that in a restricted network a set of normal nodes connected by transistors in the 1 state will attain the same target state as was shown in Corollary 6.1.1. MOSSIM simulates a transistor group by partitioning the normal nodes in a group into equivalence classes and computing the steady state signal for each class, ignoring all transistors in the X state. Then the effects of transistors in the X state are simulated by first forming a "supergraph" with a vertex for each equivalence class and an edge between two vertices if a transistor in the X state has its source node in one class and its drain node in the other. This supergraph is inspected to see which classes should be set to X because of

possible connections to classes with different state and greater or equal strength.

The MOSSIM algorithm has several drawbacks which motivated the new approach. First, the initial network analysis, as well as the computation of equivalence classes, supergraphs, and so on involves a great deal of dynamic storage allocation. This requires much computational effort and cannot be programmed easily in languages which lack automatic heap storage management. The new algorithm, in contrast, utilizes only recursive procedure calls and data structures such as sets and a small array of stacks. Furthermore, the MOSSIM algorithm cannot be extended to unrestricted networks easily. Although a set of nodes connected by transistors with strength γ_p and state 1 in an unrestricted network will form an equivalence class with the same target state, the computation of this state becomes much more difficult. While almost all MOS designs can be described as restricted networks, the greater generality of the new algorithm gives added flexibility.

7.6.2 Terman's Simulator

The algorithm used in Terman's simulator provides a timing model similar to the pseudo unit delay algorithm presented here. These two algorithms differ significantly in their functionality and internal structure, however.

Terman's program deals only with restricted networks. For nodes connected by paths of transistors in the 1 state, it combines the initial signals on the nodes with an operation similar to the operation $\vee$, just as was suggested to improve the procedure UPDATE. As mentioned in Section 2.9, however, charge sharing is simulated by real-valued capacitances. For nodes connected by transistors in the X state it attempts to encode information about the network condition into additional "states" and propagate this information much as it does the other states (which can be likened to signals.) However, the small number of additional states provides insufficient detail about the network condition, and this forces a rather inaccurate simulation. Furthermore, simply adding more states would not correct this problem, because it seems as if an accurate algorithm requires two passes over the set of nodes: the first to perform

a pre-conditioning step and the second to compute the new node states. The functionality of Tetman's algorithm can be expressed in the signal algebra (except for charge sharing) as

$$\tilde{y}_i = \begin{cases} \langle y_i(G^{on}, E^{on}) \rangle, & y_i(G^{on}, E^{on}) = y_i(G^{on}, E^{on}) \\ \lambda, & y_i(G^{on}, E^{on}) \neq y_i(G^{on}, E^{on}) \end{cases}$$

That is, a node is set to X unless it has the same steady state signal when all transistors in the X state are fully conducting as when they are nonconducting. Furthermore, when charged nodes in different states are connected by transistors in the X state, no attempt is made to take their relative capacitances into account. Instead these nodes are set to X. As was shown in Section 2.9, one has little choice in this case, because real-valued capacitances and transistor in the X state (apparently) cannot be dealt with consistently. It can be shown that Terman's algorithm is more conservative than the one given here, except when charge sharing is involved. That is, whenever it sets a node to 0 or 1 ours sets the node to the same value, but in some cases it may set a node to X when ours sets the node to 0 or 1. To see this, recall that for a restricted network, both G^{on} and G^{oo} must be elements of $\{0, \gamma_p\}^{n \times n}$, and therefore any matrix in the set $\{ G \}$ must also obey this property. Let E equal any matrix in the set $\{ E \}$ and G equal any matrix in the set $\{ G \}$. Equation 5.17 shows that $\circ$ is monotonic for arguments representing conductance matrices. Therefore for any $a \in \mathcal{A}^n$

$$x_{E^{on}} \circ x \vee y \vee x_{G^{on}} \circ a \leq x_{E} \circ x \vee y \vee x_{G} \circ a \leq x_{E^{oo}} \circ x \vee y \vee x_{G^{oo}} \circ a,$$

and furthermore, these functions are monotonic in a. A theorem similar to Theorem 6.3 then shows that

$$\kappa(G^{on}, E^{on}) \lesssim \kappa(G, E) \leq \kappa(G^{oo}, E^{oo}).$$

If $\gamma_i(G^{on}, E^{on}) = \gamma_i(G^{oo}, E^{oo})$, then $\overline{\gamma}_i(G, E) = \overline{\gamma}_i(G^{on}, E^{on})$ for any $G \in \{ G \}$ and $E \in \{ E \}$. Therefore $\tilde{\gamma}_i = \overline{\gamma}_i(G^{on}, E^{on})$. Thus, for the cases in which Terman's simulator sets a node to 0 or 1, ours sets it to the same value, except when charge sharing is involved.

Fig. 7.2. Inaccuracies in Terman's Simulator

	Initial Value	Correct Result	Simulation Result
y	0	1	X
y	1	1	X
y_1	0	X	X
y_2	1	1	X

As the examples in Figure 7.2 show, however, Terman's algorithm may set a node to X when it clearly should set it to 0 or 1. These examples are shown in the MOSSIM network model in which the pullup resistor corresponds to a d-type transistor of strength γ_1, while all other transistors are n-type transistors of strength γ_2. In the first example the node is being driven to 1 by a transistor of strength γ_1 in the 1 state and a transistor of strength γ_2 in the X state. The node will have a different steady state signal if the second transistor is conducting or nonconducting, and hence Terman's program will set it to X. The state of the steady state signal will be 1 in either case, and hence the node should be set to 1. The second example shows a similar result when a normal node is initially charged to 1, and then connected by a transistor in the X state to VDD. In the third example, if the transistor had zero conductance, node n_2 would stay charged at 1, while if it has nonzero conductance, the two nodes will share charge but n_2 will remain above the positive logic threshold. Therefore n_2 has a target state of 1, but Terman's program sets it to X. In many other cases, however, such as in simulating logic gates with some inputs in the X

Fig. 7.2. Inaccuracies in Terman's Simulator

	Initial Value	Correct Result	Simulation Result
y	0	1	X
y	1	1	X
y_1	0	X	X
y_2	1	1	X

As the examples in Figure 7.2 show, however, Terman's algorithm may set a node to X when it clearly should set it to 0 or 1. These examples are shown in the MOSSIM network model in which the pullup resistor corresponds to a d-type transistor of strength γ_1, while all other transistors are n-type transistors of strength γ_2. In the first example the node is being driven to 1 by a transistor of strength γ_1 in the 1 state and a transistor of strength γ_2 in the X state. The node will have a different steady state signal if the second transistor is conducting or nonconducting, and hence Terman's program will set it to X. The state of the steady state signal will be 1 in either case, and hence the node should be set to 1. The second example shows a similar result when a normal node is initially charged to 1, and then connected by a transistor in the X state to VDD. In the third example, if the transistor had zero conductance, node n_2 would stay charged at 1, while if it has nonzero conductance, the two nodes will share charge but n_2 will remain above the positive logic threshold. Therefore n_2 has a target state of 1, but Terman's program sets it to X. In many other cases, however, such as in simulating logic gates with some inputs in the X

As a further application of mixed-level simulation, in most MOS designs certain transistor configurations arise very often and can be replaced by their functional representations to improve the simulator speed. For example, MOSSIM recognizes the transistor configurations corresponding to 6 different nMOS logic elements: inverter, Nand, Nor, and each of these logic gates with a single pass transistor on its output. MOSSIM performs this optimization only when the configuration comprises an entire transistor group. Since transistor groups interact with one another only through fanout connections, each group has a clearly defined set of inputs and outputs. Hence the functionality of the transistor configuration can be guaranteed to match the functionality of the logic gate. For example, group G_2 in Figure 7.1 can only behave as an inverter, and group G_4 can only behave as an inverter with pass transistor. These optimizations affect only the speed of the simulation and not its functionality. This cautious approach overlooks other possible optimizations but involves no guesswork. With MOSSIM these optimizations are performed during the network analysis prior to simulation and entail little additional effort because the network must be partitioned into transistor groups anyway. With the new algorithm, no such partitioning is required unless the optimizations are to be performed, and hence the added cost of applying them becomes much higher. However, for networks which will be simulated over long sequences of inputs, the net savings can be significant.

Implementing a mixed-level simulator requires small extensions of the procedures PHASE and STEP given earlier. In addition to maintaining the list of perturbed nodes E, we must also maintain a list of perturbed function blocks F, i.e. those blocks for which some input has changed since the most recent updating. When the procedure PERTURB_TRANSISTORS is called to find which nodes are perturbed when a changing node state causes a changing transistor state, we should also call the procedure PERTURB_BLOCKS which will add any block to the list F which is perturbed by the changing node state. In the procedure STEP, blocks in the list F should be simulated and the nodes at their outputs should be set to their new states. Any node which changes state is added to the list A. With this implementation function blocks will be simulated much as they are in traditional event-driven logic gate

simulators.

7.8 Performance of MOSSIM

Although the algorithms presented in this chapter have not been implemented, their performance should be comparable to MOSSIM. Thus we can use the performance of MOSSIM as a measure of the speed of switch-level simulation. Furthermore, since MOSSIM can replace transistor configurations corresponding to certain logic gates by a gate-level representation, we can compare the relative speeds of switch-level and logic gate simulation.

MOSSIM is written in the language, CLU [26]. All times were measured on a DEC 20/60. While the program, programming language, and computer system have been designed to provide reasonable performance, there is room for speed improvements in all three areas.

Table 7.I lists the characteristics of six different binary counter circuits, three each of two basic designs. Both designs have the circuit shown in Figure 7.3 for each bit position. Data is stored dynamically, and no initialization circuitry has been included. The chain of half adders forms a carry-ripple adder. The two designs differ in how the half adders are implemented. The first, called CNTR, utilizes four Nand gates and an inverter to implement the sum and carry logic in a conventional way. The second, called MCNTR utilizes a pre-charged Manchester carry chain as shown in Mead and

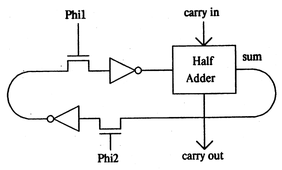
---

Program 7.I. Test Case Networks

Name	Transistors*	Logic Gates
CNTR10-OPT	0	70
CNTR10-UNOPT	200	0
CNTR16-OPT	0	112
MCNTR10-OPT	124	52
MCNTR10-UNOPT	258	0
MCNTR16-OPT	200	84

* includes depletion mode transistors

Fig. 7.3. One Bit of Counter Circuit



Conway [28, p.150] to implement the carry logic and a selector logic block [28, p.152] to implement the sum logic. This design achieves high speed by pre-charging each bit position in the carry chain on each clock cycle, so that the carry lines are never driven through load transistors.

Both designs were tried in 10 and 16 bit versions (e.g. CNTR10 and CNTR16). The suffix "OPT" for the entries in Table 7.I indicates that MOSSIM replaced the transistors for whatever logic gates it could find with the logic gate representation. The suffix "UNOPT" indicates that the network was simulated entirely at a transistor level. As can be seen, the conventional counter design can be replaced entirely by a gate level representation, while in the other design only 50% of the network (measured by the number of transistors) could be replaced. Experience has shown that between 50% and 80% of typical designs can be represented at a gate level.

Table 7.II gives performance data for the six circuits. All times are measured in CPU milliseconds per clock cycle. Best and worst case times were measured by finding which cases minimized or maximized the time, while average times were measured by averaging 1024 clock cycles. The best and worst case times could not be measured with complete accuracy for the faster circuits.

Program 7.II. Performance Statistics

CPU milliseconds / clock cycle simulated

Circuit	average	best	worst
CNTR10-OPT	30	30	70
CNTR10-UNOPT	95	70	220
CNTR16-OPT	34	30	110
MCNTR10-OPT	300	260	320
MCNTR10-UNOPT	400	360	440
MCNTR16-OPT	490	450	530

First let us consider the data for CNTR. The transistor level simulation requires 3 times longer than the gate level simulation. This provides a measure of speed of a simulator which operates at a transistor level relative to one which operates at a logic gate level. Furthermore, if the simulator were designed only to simulate logic gate circuits, its speed could be improved further. Nonetheless, it shows that the speed of a switch-level simulator can approach that of a logic gate simulator. Observe that the best and worst case times differ greatly. This indicates that unless the carry propagates a long way during a clock cycle, large portions of the network remain inactive. This is also seen in the 16 bit version where the average time is only slightly higher than for the 10 bit version, but the worst case time grows in proportion to the network size.

The data for MCNTR indicate much different performance characteristics for this design than for a conventional gate implementation. The circuit requires up to 13 times longer (for the 16 bit version) than CNTR. This difference is due largely to the greater amount of activity in MCNTR. On every clock cycle, all carry lines are pre-charged, and since the sum logic depends on the carry values, the simulator will first compute the sum based on the pre-charged value and then on the final carry value. Furthermore, transient X states arise due to sneak paths in the push-pull drivers for the carry chain and the sum logic, causing many more activities to be simulated. Thus, the amount of activity in the network is almost independent of the length of the carry propagate. This is indicated by the closeness of the best and worst case times and by the fact that the simulation time grows in direct proportion to the network size. Note

also that in replacing 50% of the network with logic gate representations, we improve the speed by 25%, showing that these optimizations do not provide a major performance gain.

These measurements indicate that the performance of a switch-level simulator can vary widely depending on the nature of the circuit to be simulated. For circuits implemented mostly with logic gates, the activity is highly localized and much of the design can be simulated at a gate level. For designs using more exotic techniques such as pre-charged and pass transistor logic, activity occurs throughout the circuit and a larger portion must be simulated at the transistor level. In either case, however, the simulation time grows at most linearly with the network size.

7.9 Summary

Unlike previous switch-level simulators which were based solely on the intuitions of the designers, the algorithms presented here are based on a mathematical theory. This provides a framework much like numerical analysts have in which problems are formulated as a set of equations, and the goal becomes to find efficient algorithms to solve them. In our case, the simulation algorithm relies mainly on a technique for solving recurrence equations in the strength algebra. By studying this simplified and more abstract problem, one can see more clearly the trade-offs between algorithmic complexity and simplicity of implementation. Furthermore, the algorithms can be proved correct. The characteristics of equations which arise in simulating MOS networks such as sparseness and locality of changes can be exploited to obtain an algorithm which is particularly efficient for this application.

Once the basic unit delay algorithm has been developed, it can be altered to provide a slightly different timing algorithm or extended to improve the efficiency and generality of the simulator. It can be seen that switch-level simulation can be combined with logic gate and functional simulation so that the best features of each may be utilized.

8. Timing Models

8.1 Introduction

The unit delay algorithm presented in Chapter 7 simulates the network as moving through a sequence of target states. To the external viewer, this provides a timing model in which a transistor switches one time unit after its gate node changes state, but in which signals propagate along paths of conducting transistors instantaneously. For common implementations of inverters, Nand gates, and Nor gates, such as those shown in Figure 2.2, this algorithm also simulates logic gates as having unit delay. In this chapter, the modeling of time in MOS circuits will be investigated in terms of both simulating the functional behavior of a design and detecting timing errors. Example networks will be given containing logic gates. It is assumed that these gates are implemented in one of the styles shown in Figure 2.2, all of which behave identically from a logical point of view.

As was mentioned in Chapter 2, switch-level networks share many commonalities with relay and logic gate networks when viewed as systems computing logical functions. The target state provides the basic characterization of the logical function computed by a switch-level network. It gives the node states created by the network in response to the current states. Thus it describes the *excitation* of the network much as the excitation of a Boolean gate network [20] is defined as the output values of all logic gates as functions of their current input values. Many of the theoretical techniques and algorithms developed for logic gate networks (and relay networks) can be adapted to the switch-level model. In doing so, however, several characteristics of MOS systems must be kept in mind.

First, the sheer size of MOS LSI systems imposes constraints on the practicality and usefulness of many techniques. Such tools as Karnaugh maps [20] and flow tables [22, 43] require a complete enumeration of all possible network states, which would grow exponentially with the number of nodes. Such techniques can only be applied to small sections of a design. In fact, any algorithm with nonlinear time complexity must be viewed with skepticism for networks in which the elements number in the

thousands. Similarly, any tool which requires "hints" from the user such as the location of feedback paths, state variables, or delays becomes unwieldy for large networks, unless this information can be derived algorithmically from some source such as the layout specification. This problem becomes even greater as LSI designs are generated by automated or partially automated systems, because the designer may not have the intimate knowledge of the network required to provide such hints. Thus the desire to handle large networks and to implement any technique as a computer algorithm changes the standards by which techniques are measured.

As a further point, switch-level networks differ from networks in other logic models in that the X level can arise during normal network operation due either to short circuits or improper charge sharing. For example, sneak paths between input nodes in different states often arise in pass transistor logic circuits such as the push-pull drivers used in output pads [28, p. 165]. Generally these error conditions occur transiently and have no effect on the ultimate network behavior. The presence of X states may not indicate a badly designed circuit but only a temporary ambiguity in the network operation which must be scrutinized to see how far it propagates. This contrasts with logic gate models in which either Boolean behavior is assumed at all times or at least that an X value can only arise as the result of other X's. Techniques developed for other logic models often require modification to handle the X state.

Traditional assumptions about sources of delay in digital systems do not apply to MOS circuits, either. Most analytic techniques for asynchronous circuits [22, 29, 43], assume that delays occur only in logic elements (gate delay) and not in wires (line delay.) Furthermore, they assume that a logic element will respond to all inputs simultaneously. This would require in MOS implementations of logic gates, such as those shown in Figure 2.2, that all transistors respond to changing input signals at the same rate. Such an assumption may not hold, because p-type and n-type devices may have different timing characteristics, and even transistors of the same type may not behave identically due to variations in geometry, fabrication details, and stray capacitances. A timing model for MOS circuits should allow each transistor to have an independent switching time. With this degree of generality, however, we can also

model line delay by incorporating it into the switching times of the transistors with their gates attached to the wire. Furthermore, transit delay (the time required for a signal to travel through a conducting transistor) can usually be modeled by incorporating it into the switching time of the transistor. Thus, a timing model which allows arbitrary delay times for each transistor can model all forms of delays in an MOS circuit. Given that wiring delays are predicted to dominate in future VLSI circuits [36], the ability to model wiring delays may play an important role. Many of the traditional analytic tools, however, cannot deal with this degree of generality.

As a final, and somewhat more optimistic note, much of the concern about timing in traditional logic design need not concern the MOS designer. Most MOS systems are designed to operate synchronously with conservative clocking schemes. For example, in a properly designed circuit with a two-phase, nonoverlapping clocking scheme [28], no malfunctions due to timing can arise as long as the clocks run slowly enough for the circuit to settle during each time epoch, but fast enough to avoid the loss of stored charge. These methodologies have been adopted, in fact, to compensate for the difficulty in accurately predicting the exact time behavior of MOS circuits, especially if the design is to operate correctly despite variations in fabrication and despite the inability to fine tune circuits once they have been fabricated. For such systems, almost any timing model would provide sufficient accuracy to test the functionality of a design. However, timing still remains an issue for those designs in which relative path delays can affect the logical behavior and for ascertaining that a design can operate at a particular clocking rate.

8.2 Simulation Timing Models

Much attention has been focused on timing models for logic gate simulators. Some of these techniques will be described and their suitability for switch-level simulation will be discussed.

8.2.1 Unit Delay

To implement a unit delay model the simulator computes the excitation of the network and then sets the node states to these values. This model has been used successfully in both logic gate and switch-level [9] simulators. It provides the same level of accuracy as logic designers use when they analyze timing by counting logic gate delays. This suffices to model the functionality of a a wide variety of designs, because very few circuits rely on a path with fewer logic gates having a longer delay than a path with more. A unit delay model, however, may deceive the logic designer who finds that a design can be made to simulate correctly if extra delays (e.g. inverter pairs) are inserted along some paths. Often the actual circuit cannot be corrected so easily because of factors such as the assymetric rise and fall times of ratioed logic gates and the inexact behavior of the circuit during transitions.

Unit delay simulators can fail to terminate, both in cases where the actual circuit would run indefinitely, and in cases where the actual circuit would settle. For example, a simulation of an inverter ring would not terminate, such as the one formed when the input to the network shown in Figure 8.1a is set to 1, because the circuit would run indefinitely. More importantly, the circuit shown in Figure 8.1b has a critical race if the input is changed to 1, but eventually the slight differences in the two Nand gate delays would cause the conflict to be resolved (although not with a predictable outcome.) With a unit delay simulator, however, both logic gates are simulated as having the exact same delay and the oscillations continue indefinitely. As a practical matter, this problem has arisen only a few times out of many simulation runs by many users. The conditions leading to these oscillations seldom appear in real designs. This nontermination due to perfectly matched delays can create major difficulties, however, if

Fig. 8.1. Networks for Which Simulation May Not Terminate

the simulator tries to initialize the network nodes to states other than X, because a naive choice could well create effects similar to the example in Figure 8.1b. To prevent nonterminating simulations, the simulation program can be designed to halt after a maximum number of unit steps, with this limit set according to the network size.

8.2.2 Pseudo Unit Delay Simulation

A pseudo unit delay simulator proceeds by computing the excitation of a set of nodes resulting from an *event* selected from an event list, where each event indicates a perturbation in the network state. These nodes are then set to their excitations and any resulting perturbations are added as events to the end of the event list. This process continues until the event list is empty, indicating that the network is in a stable state. As long as the event list is maintained as a first-in, first-out queue, this simulator resembles a unit delay simulator in that if two nodes are perturbed simultaneously, the effects of both will be simulated before any perturbations they cause are simulated. However, activities which are modeled as occurring simultaneously (and hence independently) in a unit delay simulator will be ordered, and hence one may affect the other. This ordering will depend on the internal details of the simulation program and to the user will appear unpredictable. Such a simulator would terminate for the example shown in Figure 8.1b, although not in a predictable way, because the simulator would arbitrarily select one logic gate to simulate before the other. The network shown in Figure 8.1c, however, has a similar form of critical race, but a pseudo unit delay simulator would fail to terminate, because in following a FIFO discipline the simulator would alternate between the upper and lower chains, giving the effect of perfectly matched delays. One should note, however, that this example is very contrived, whereas networks like that of Figure 8.1b are quite common. It is not known whether less pathological examples would fail to terminate with a pseudo unit delay simulation in cases where the actual circuit would reach a stable state. Circuits with nonterminating behavior such as the inverter ring shown in Figure 8.1a will have nonterminating simulations. Thus, a pseudo unit delay simulator partially solves the problem of unbounded oscillations,

but at the expense of introducing some degree of unpredictability. This technique has been used successfully in a switch-level simulator [5].

8.2.3 Zero Delay

To implement a zero delay model [12] all feedback paths must be broken such that the system becomes a combinational network, computing a function from the inputs and current values of the state variables to the outputs and new values of the state variables. Each pass through the network appears to occur instantaneously, and hence the term "zero delay". This model assumes the circuit is free of critical races. With this approach an ordering can be placed on the network elements such that each element is simulated at most once during a pass, thereby achieving greater efficiency than either a unit delay or pseudo unit delay simulator. Simulations of networks such as those shown in Figures 8.1b and 8.1c would terminate, assuming that the paths in both cases are broken in only one place, although a simulation of the network shown in Figure 8.1a would not. Such a technique would apply to MOS networks only if the feedback loops could be identified automatically. Finding a minimum set of feedback loops is known to be an NP-complete problem [17], but for most applications a set which is not minimum would suffice. If an algorithm chooses to break the paths in Figures 8.1b and 8.1c in two places, however, a nonterminating simulation could result. Furthermore, the user would have little understanding of the simulation timing unless informed of the points at which feedback paths are broken. A zero delay model has apparently not been tried in a switch-level simulator.

8.2.4 Continuous Time

Many logic gate simulators introduce a continuous time measure by allowing the user to assign delay times to the logic elements and using a time-ordered event list scheduler. Some even allow logic gates to have different rise and fall times [44], to model logic gates in ratioed circuits. Such a technique has been proposed for switch-level simulation with the network parameters which determine the delay computed by a layout analyzer [5]. With MOS circuits, however, delays are affected by many details which the switch-level model ignores, such as both the linear and nonlinear effects of transistors, the threshold voltages, the capacitive loadings (which may change during operation), and the exact voltage waveforms on the nodes. As one tries to take such details into account, it becomes difficult to achieve a consistent level of detail without resorting to a full scale analog simulation. An inaccurate simulation would create more problems than it would solve, because users tend to place great faith in numerical results even if they have no validity.

Perhaps the most promising approach is to find lower and upper bounds on the circuit delay by applying only simplifications of the analog behavior which can be guaranteed either conservative or optimistic. The user can then tighten the bounds by increasing the level of detail at which the circuit is simulated, until it can shown that the required timing conditions will be met. Recent work by Glasser [18] takes important steps in finding these kinds of simplifications, but much more work is required before it becomes a practical simulation tool. This form of simulation would provide the most reliable verification of the ability of a circuit to operate at a particular clocking rate.

8.2.5 Summary

None of the models listed above will satisfy all users at all times. However, the unit delay and pseudo unit delay models stand out for their simplicity, understandability, and consistency with the level of detail which the switch-level model is intended to provide. To choose between these two, one must compare the value of greater predictability against the value of a greater (but not total) immunity to nonterminating simulations.

8.3 Analysis of Timing by Ternary Simulation

Rather than introducing a continuous time model into a switch-level simulator, one might best leave such a level of detail to analog simulators where it can be handled accurately. Instead, the switch-level simulator could be used to identify those portions of a system which warrant closer timing analysis by analog simulation or some other technique. This would allow the more powerful (but more expensive) tools to be applied just where they are needed. For example, the speed of a synchronous circuit is often limited by a single critical path, such as the carry chain of an adder. A unit delay simulator could generally find this path by finding which nodes changed state during the last unit step of a simulation phase and then working backward.

A method known as *ternary simulation* has been developed to detect possible hazards and critical races in logic gate circuits without introducing a continuous time model [Ji, 23, 46]. This technique uses the X state to represent the ambiguity caused by a node in transition from 0 to 1 or 1 to 0. Ternary simulation techniques can also be applied to switch-level networks to detect possible sources of timing errors. These parts of the circuit can then either be redesigned or analyzed by more detailed methods such as circuit simulation.

8.3.1 Algorithm

The algorithm for a ternary simulation of MOS networks can be described in terms of the function *phase*. Suppose a switch-level network is in a stable state $\mathbf{y}$, i.e. $\mathbf{y} = step_{\mathbf{x}}(\mathbf{y})$, and we wish to simulate the effects of changing the input nodes to state $\mathbf{x}'$. The resulting state $\mathbf{y}'$ is computed as

$$\begin{aligned} \mathbf{q} &\leftarrow phase(\mathbf{x} \sqcup \mathbf{x}', \mathbf{y}) \\ \mathbf{y}' &\leftarrow phase(\mathbf{x}', \mathbf{q}). \end{aligned}$$

That is, first all nodes i_j for which $x_j \neq x'_j$ are set to X, and the network is simulated until it settles. Then the input nodes are set to their final values $\mathbf{x}'$ and the network is again simulated until it settles. For logic gate networks implemented with transistor circuits such as those shown in Figure 2.2, this algorithm reduces to the one proposed by Brzozowski and Yoeli [11] for logic gate networks. Ternary simulation requires only slightly more effort than the unit delay simulator described earlier. Observe, however, that the X state will now become the most prevalent state during the simulation, because every time a node makes a transition, it must go through the state X. For the method to provide useful information, the X state must be simulated accurately and efficiently. The algorithms presented in Chapter 7 satisfy these requirements.

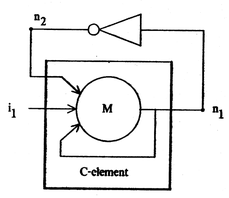
It is claimed that each node n_i will have a new state y'_i equal to 0 or 1 if and only if it would have this unique state regardless of the magnitude of any delays in the circuit. Any nodes sensitive to network delays will have y'_i equal to X. This claim has not been proved formally, although it can be motivated informally, and weaker statements have been proved.

The ternary simulation method will first be motivated informally. In setting the input nodes to $\mathbf{x} \sqcup \mathbf{x}'$, all inputs which may be changing simultaneously are set to X, indicating that their exact values cannot be ascertained. In the first computation of *phase*, these X's are propagated to all nodes which are sensitive to the input node transitions, including into the feedback paths corresponding to unstable state variables. In the second computation of *phase*, the final values of the input nodes are propagated to any

nodes which are combinational functions of the inputs and of any stable state variables. If a feedback loop has developed containing all X's, however, these nodes and any nodes dependent on them will remain in the X state, because the behavior of the actual circuit would depend on the exact delays in the feedback paths. The assumption made in our logic model that a set of transistors with the same gate node may behave independently when the gate node is in the X state gives the effect of each transistor responding to a transition at an independent rate. Thus a node will be set to 0 or 1 if and only if it has this unique state regardless of the transistor switching delays in the circuit.

For example, the network shown in Figure 8.2 contains a 2-out-of-3 majority with its output fed directly to one input and through an inverter to another. An MOS implementation of the majority gate is shown by Seitz in Mead and Conway [37, p.255]. The output of this gate equals 1 or 0 if at least two inputs equal 1 or 0, respectively, and equals X otherwise. The reader can also verify that an MOS inverter in our model will have output X if its input equals X. Suppose initially that $x_1 = 0$, $y_1 = 0$, $y_2 = 1$, and that we change x_1 from 0 to 1. Assuming unbounded transistor switching delays (or equivalently unbounded line delays), there will be a critical race depending on the relative delays in the two feedback paths. A ternary simulation would first set x_1 to X, and then compute the new state X for

Fig. 8.2. Ternary Simulation Example



both nodes n_1 and n_2. Thus, feedback loops have developed containing only X's. When x_1 is then set to 1, both n_1 and n_2 will remain at X, indicating a critical race. This example demonstrates that a Muller C-element (consisting of a majority gate with its output fed back to one of its inputs) implemented in MOS may malfunction if its feedback path contains an excessive delay. In fact, Dennis and Patil have shown [14] that a C-element cannot be constructed which is guaranteed to function despite arbitrary line delays.

8.3.2 Theoretical Results

Brzozowski and Yoeli [11] have proved that an algorithm similar to the one shown here will set a node in a logic gate network to 0 or 1 *only if* it has this unique state regardless of *logic gate* delays. Some networks which would function properly assuming only unbounded logic gate delays, such as the example of Figure 8.2, however, may have nodes set to X. These authors conjecture, but do not prove, that their algorithm will set a node to 0 or 1 *if and only if* it has this unique state regardless of both *gate* and *line* delays. Their proof relies primarily on the monotonicity of the excitation function for the partial ordering $0 \leq X$ and $1 \leq X$. Their proof also assumes that a logic network normally contains only nodes with Boolean logic states, but this requirement can be relaxed. The excitation function for the switch-level model is $step_X$, and we have already seen by Theorem 6.6 that it is monotonic. Thus, Brzozowski and Yoeli's proof can be applied to show that for y' resulting from the ternary simulation, each element y'_i will equal 0 or 1 only if it has this unique state regardless of the transistor switching delays, with the restriction that transistors with the same gate node must have the same delay. Furthermore, a proof of their conjecture could be applied to show that each element y'_i will equal 0 or 1 if and only if it has this unique state for arbitrary transistor switching delays.

Brzozowski and Yoeli's proof also shows that the ternary algorithm will always terminate after a linear number of simulation steps (in the number of nodes.) For example, the simulations of all three networks shown in Figure 8.1 will terminate, (but with the nodes in the X state.) Even though such circuits as the inverter ring shown in Figure 8.1a will run indefinitely, their ternary simulations will terminate. Thus, the worst case performance of ternary simulation is better than that of any of the timing models described previously. In practice, this potential gain is offset by the larger number of node transitions and the greater effort required to simulate the effects of the X state.

8.3.3 Application

A ternary simulation would provide useful results for synchronous systems, because a well designed synchronous system should contain no critical races under any delay model. Timing errors can only arise as a result of insufficient clock intervals, and these should be checked by critical path analysis. For asynchronous systems, on the other hand, Dennis and Patil [14] have shown that no nontrivial sequential circuits can be built which will function correctly despite arbitrary line delays. For example, the Muller C-element forms the cornerstone of most speed independent circuit designs [29], and we have already seen how it can fail with ternary simulation. A ternary simulation of most asynchronous circuits would overwhelm the user with X's and provide little information about the design. Unger [43] describes how delay elements could be introduced into a ternary simulation. By inserting delay elements into carefully selected feedback paths, circuits can be made to simulate in a reasonable manner. This style of asynchronous design, however, has limited application in MOS LSI, because it requires too much fine-tuning of the circuit.

8.4 Toward a Simulator for Self-Timed Systems

In some design disciplines, timing constraints are assumed to hold within small regions of the circuit, but as long as these constraints are satisfied the system will function correctly regardless of the delays in other parts of the circuit. For example, many speed-independent designs require only that the C-elements operate properly. Seitz has proposed a class of systems called *self-timed systems* [36] in which particular timing constraints can only be assumed to hold within *equipotential regions*, while arbitrary delays may be incurred by any communication between regions. This methodology allows much flexibility in the design style, because a subsystem contained within a single equipotential region can be implemented with a synchronous or asynchronous circuit, or recursively as a self-timed system. Self-timed systems may also have overlapping equipotential regions, but we will not consider this possibility. Ternary simulation would provide a useful tool for testing self-timed systems if it were applied only to those portions which are to function correctly despite arbitrary network delays. The simulator should only model the functionality of those portions for which particular timing conditions are assumed to hold, thereby providing the appropriate stimulus to the portions simulated by the ternary algorithm.

Such a simulator would have many advantages over detecting timing errors with a continuous time model simulator. With a continuous time simulator, numerous cases must be simulated with slight changes in operating conditions and network parameters. Even a poorly designed circuit may simulate acceptably due to a chance combination of input conditions, element parameters, and simulation model. With ternary simulation all possible forms of uncertainty due to timing are condensed into a single state and hence a single simulation run analyzes many cases simultaneously, always finding the worst case behavior. With a self-timed system simulator, the user could isolate small regions of the design for which particular time behavior is assumed to hold. These regions could be tested extensively be a very accurate circuit simulation. Then the simulator tests the hypothesis that as long as these regions function correctly,

the entire design will be insensitive to other circuit delays.

A system could be simulated in this manner if the regions of the circuit for which ternary simulation is to be applied were separated from those regions to be modeled with a pseudo unit delay simulation by special network elements as will be described informally. These elements serve to convert between the two modes of simulation. Regions may only be connected through fanout connections, i.e. from a node in one region to the gate of a transistor in another.

Ramp elements convert the 0-1 and 1-0 transitions resulting from a pseudo unit delay simulation to 0-X-1 and 1-X-0 transitions for the ternary simulation. When the input to a ramp element changes, the output is first set to X, initiating a *transition event*, an indication of which is placed on a special event list. Any time a perturbation resulting from a transition event causes an X to propagate to a node, this change also becomes a transition event. Events are selected from the normal event list only when the transition event list is empty. The simulation is continued until the network settles (i.e. both event lists are emptied), at which time the X's will have propagated as far as possible. Then the simulator sets the outputs of those ramp elements with output equal to X to their input values, and the effects of these events are simulated until the network settles. It can be seen that the combination of ramp elements and this scheduling algorithm leads to a ternary simulation of those regions with ramp elements at their inputs.

Trigger Elements convert the 0-X-1 and 1-X-0 transitions resulting from a ternary simulation into 0-1 and 1-0 transitions. That is, if a transition event causes the input to a trigger element to be set to X, the output is held in its previous state. When the input is set to a new state by a normal event, the output is set to this state. In this way, the X states arising from the ternary simulation will be blocked from entering regions for which the simulator is only to model the functionality.

While this simulation technique has not yet been implemented, it shows great promise as a tool for locating possible timing errors in self-timed systems.

8.5 Summary

The relatively simple unit delay and pseudo unit delay timing models have proved adequate for modeling the functional behavior of both synchronous and self-timed systems. These models provide little information about possible timing errors, but for circuits designed with conservative clocking schemes, such information is not required. A continuous time model simulator would prove most useful for verifying that a circuit can sustain a particular clocking rate. Such a simulator should try to place lower and upper bounds on circuit performance by applying only simplifications which can be guaranteed conservative or optimistic. It should allow the user to tighten these bounds by increasing the level of detail at which the circuit is modeled. Ternary and self-timed system simulators provide a powerful tool for detecting critical races. Unlike analog simulators, which can only verify a design for a particular set of circuit delays, these simulators can verify a design for all possible sets of circuit delays, except for small regions in which particular timing conditions must be assumed to hold.

9. Conclusions

9.1 Final Thoughts

The value of the switch-level model has already been proved by extensive experience with switch-level simulators. These simulators have shown great versatility and accuracy in modeling the logical behavior of systems constructed using many different architectural and circuit design techniques. Furthermore, they utilize only information about the actual structure of the design such as can be obtained from the mask specifications. By operating at a logical level, switch-level simulators achieve performances approaching those of logic gate simulators.

This thesis has demonstrated that the switch-level model can also be developed into a mathematical description of the behavior of MOS logic circuits. This allows a rigorous derivation of equations to express the operation of a network from which simulation algorithms can be derived and proved correct. These new algorithms improve on previous algorithms which were based on the programmer's intuition.

The degree of generality allowed by the switch-level model does not come without its costs. As compared to relay and logic gate models, our mathematical model seems much more complex and intractable. A long and difficult process was required to derive the basic simulation technique. This difficulty stems partly from the novelty of the model. The switch-level model cannot be described adequately by such well-developed concepts as Boolean and linear algebras, and hence we were forced to develop a new abstraction and justify it in terms of an electrical model. Furthermore, the switch-level model supports a much richer variety of circuit and architectural design techniques than do traditional logic models. The Mead and Conway approach to custom LSI design differs from other approaches such as polycells [33] and gate arrays [19] in that it allows the designer to select from the entire variety of different design techniques and even tailor the individual devices to provide many trade-offs between speed, power, density, and ease of design. To provide sufficient expressive power for this level of generality, a logic model must be more complex than logic gate models, but as has been shown this

generality need not come through *ad hoc* extensions.

Unlike commercial LSI designs, however, Mead and Conway encourage the use of structured design methodologies and simplified design techniques. This permits a heavy reliance on computerized design and analysis tools to replace the many hours of highly skilled labor used to produce commercial LSI designs. The tools can be designed along the lines of this methodology, thereby achieving better performance and encouraging the user to produce well structured designs. With the emphasis shifting away from techniques appropriate for humans to those appropriate for computer implementation, techniques should be judged primarily on their ability to be implemented as computerized tools. In this regard switch-level simulators compare favorably with logic gate simulators and greatly outperform analog simulators.

9.2 Suggestions for Further Research

Thus far, the switch-level model has only been implemented in the form of logic simulators with simple timing models. While these applications have demonstrated the value of the switch-level model, they represent only the beginnings of an entire class of tools for MOS design. The switch-level model helps bridge the gap between the views of an LSI design as an electrical circuit or as a piece of artwork and the view as a system which computes a logical function.

9.2.1 Simulation

Simulation will always play an important role in the LSI design process. Humans have a remarkable ability to synthesize designs, often applying great cleverness in selecting from a seemingly endless range of possible approaches. However, many hours of tedious work and much self-discipline are required to generate a totally correct design by hand. Computers, in contrast, lack the kind of cleverness and intuition which goes into novel and original designs but will willingly perform tasks which humans find impossibly dull. Thus a natural complement is formed with computers verifying the designs and

ideas produced by humans. Simulation provides just that form of verification. It directly tests the functionality of a design in much less time and at much less cost than the production of prototypes. Furthermore, it can often provide more information than can reasonably be measured from an actual implementation of a design.

Future logic simulators could provide more rigorous checks of the correctness of a design. Unlike current simulators, which try to provide as much generality as possible, the simulator could be tailored toward a particular design methodology to check whether the design adheres to this methodology. Many of these checks can be added with little additional complexity.

A slight modification of the algorithm given in Chapter 7 would yield a simulator which provides a more rigorous test of CMOS designs. In CMOS, node voltages correspond to valid logic levels only if they very nearly equal 0.0 or V_{dd}. Even the threshold voltage drop resulting from a signal with state 1 passing through an n-type transistor or a signal with state 0 passing through a p-type transistor is considered excessive. Thus, a 1 signal passing through a conducting n-type transistor should become an X and similarly for a 0 through a p-type, unless the complementary transistor is connected in parallel. A simulator can achieve this effect by modeling a turned-on n-type transistor as having conductance equal to its strength for signals of state 0 and having an unreliable conductance for signals of state 1, and conversely for p-type transistors. The techniques developed for modeling transistors in the X state can be extended to describe the behavior of the network for this model with only slightly increased effort.

The self-timed system simulator described in Chapter 8 would test whether a circuit really fulfills the requirements of self-timed system design. It does so in a more rigorous and reliable way than would an analog simulator and with much less computational cost. The algorithm for this simulator must still be developed in detail, however, and the design of a suitable user interface will present many challenges. A more formal understanding of the theory behind this simulation method is also required. Brzozowski and Yoeli's conjecture that a ternary simulation tests whether a design will function properly for arbitrary line delays remains an open problem. The proposed combination of ternary and pseudo unit delay

techniques has also not been studied formally. This style of simulation could greatly assist the design of self-timed systems.

Simulators could also be designed to provide new kinds of information. For example, whereas existing simulators only tell the user the state of the network, a more sophisticated program could also tell how it got there. This would greatly aid the user in debugging a design.

Simulators could also provide more information about circuit timing. Even with methodologies such as conservative clocking schemes and tools such as ternary simulation, for some applications the user must know the time behavior of a circuit. For example, if a system must function at a clock rate determined by other chips or by the particular application, the designer must verify that this clock rate can be sustained. Hopefully, a full scale analog simulation can be avoided by identifying the critical path and simulating only this part with a simplified model. It is believed that a considerable amount of detail must be added to the switch-level model before it can model circuit timing with the same generality and accuracy with which it models the functionality. While the "order of magnitude" electrical model may describe the logical behavior adequately, it provides limited information about circuit speed. Ideally, any simplifications of the electrical behavior should be guaranteed conservative or optimistic so that the simulator provides bounds on the performance. The simulator should be able to apply different levels of detail so that the bounds can be tightened until they are acceptable for the application. Unlike existing hybrid simulators in which increased accuracy is achieved only by going to a totally different (and often incompatible) model, the simulator should allow a smooth and reliable transition between the levels of detail.

A switch-level simulator could also provide information about the effects of circuit faults. Unlike the traditional methods of fault analysis, which assume that a logic gate will fail by becoming "stuck-at" some level, the switch-level model can describe the failure of individual transistors or connections. For example, a faulty transistor can be modeled as a transistor in the X state, i.e. an unknown and unreliable conductance. Similarly, a faulty wire can be modeled as two nodes connected by a strong transistor in the

X state. This application of the X state has not been studied in detail.

Finally, we must consider what form of simulation will work for VLSI design. The current implementations of switch-level simulators model the entire design as a network of transistors or at best combine some of these transistors into simple logic gates. While the efficiency of this technique has proved adequate for modeling LSI systems containing up to 10,000 transistors, it will clearly fail for VLSI systems containing 100,000 or perhaps 1 million transistors. As with all other design tools, the size and complexity of VLSI systems forces a rethinking about logic simulation. Systems of this size must be designed hierarchically, where each level of the design involves combining subsystems designed at the next lower level. A simulator must utilize this hierarchy to achieve the required performance and to assist the user in maintaining the hierarchy. Switch-level simulation can provide valuable assistance at the lower levels of this hierarchy. It can be used to verify that a transistor circuit implements a particular logical function either by automatically comparing it to a functional specification or simply by allowing the user to test the design. Once the functionality has been verified, the simulator can replace the switch-level representation with a functional representation for simulation at the next level of the hierarchy. In some cases the user may also wish to simulate some portions of the network at a functional level and other portions at a switch level. As was seen earlier, a simulator can be designed to combine simulations at these different levels. Thus switch-level simulation will serve an important role in VLSI design.

9.2.2 Other Design Tools

Most existing LSI design tools treat a design as a piece of artwork. For example, most layout systems and design rule checkers view the design as a collection of geometric primitives with no understanding of the actual or intended functionality. As a consequence, they provide only limited assistance to the designer. The switch-level model can help introduce a greater understanding of the functionality of a design.

Current design rule checkers, for example, check only the geometric features in the layout. Experience indicates that they catch only a limited class of errors and also report many extraneous errors. For example, if the designer omits a contact cut, no error is reported. If a wire contains a gap, it will be reported only if the gap is smaller than the interwire spacing tolerance. A more useful checker would perform both the geometric analysis and the extraction of the switch-level logic network. It would produce two results: the switch-level network it believes was intended, and the design rules violated by the layout with respect to the network. By simulating the network and checking the list of violations, a designer would have a much greater chance of detecting most errors. Furthermore, since the checker would have a better understanding of the electrical connectivity in the layout, it could greatly reduce the number of extraneous errors reported. Experience with C. Baker's layout extraction program [4, 5] has already demonstrated that many errors in the artwork can be detected by deriving the switch-level network and applying several simple checks.

The switch-level logic model could also be incorporated into a design "analyzer" which detects such features in an MOS logic design as feedback paths, sites of dynamic storage, potential race conditions and short circuits. Many of the theoretical techniques which have been developed for the Boolean gate model could be profitably adapted to the switch-level model.

The ultimate goal of computer-aided design is to automatically synthesize an LSI or VLSI design from a high level functional description of the system. Current automated design systems such as Bristle Blocks [24] still require the user to design large portions of the layout by hand and have only an incomplete understanding of the circuit's functionality. These tools would benefit to some degree by adopting the uniform representation of a logic design provided by the switch-level logic model.

9.2.3 Theoretical Developments

The mathematical developments in this thesis provide a firm foundation for switch-level simulation, but much more theoretical work is required before the full potential of the switch-level model can be realized. Several areas have already been mentioned in connection with their possible applications. For example, the exact relation between circuit timing models and both ternary and self-timed system simulation techniques remains to be established. Similarly, research has only begun on methods for introducing more detailed timing information in ways which can be guaranteed conservative or optimistic. Finally, developing computer algorithms to identify feedback paths, sites of dynamic storage, and other features of a design will require a much better theoretical understanding of the structure of MOS logic networks. Many of these analysis problems appear to be NP-complete, but heuristic techniques can probably be devised which will work efficiently for most real designs.

The simulation model developed in this thesis provides only an operational model of a logic network. That is, given a particular initial state, it describes the state in which the network will ultimately stabilize. A more powerful model would describe the function computed by the logic network. For example, Shannon showed how the Boolean function computed by a combinational relay network can be derived from the structure of the network. The Boolean function for a logic gate network can generally be derived by inspection. Work on denotational semantics of programming languages is also directed toward deriving the function computed by a computer program. Much more work will be required to develop a corresponding functional model of switch-level networks. At present, the conversions between the logic states, logic signals, and signal strengths tend to obscure the function computed by the network. Furthermore, expressing the ternary function computed by a network (i.e. including the state X) involves considerably more effort than expressing the Boolean function.

A functional model of switch-level networks would have many applications. For example, it could be used to verify that a network implements a particular function. Unlike programming language models in which this task is in general uncomputable, for logic networks we can always construct truth tables by simulation, and hence any design can be verified in at most exponential time. Furthermore, various heuristic techniques could reduce the complexity considerably for most problems. Therefore, automatic logic design verification may be practical. A functional model could also help make the transition from a switch-level simulation to a functional level simulation. That is, if the function computed by a section of a design could be derived automatically, the simulator could replace the switch-level representation by a functional representation. This seems especially practical if it is applied to small sections of the design, such as to the transistor groups used in MOSSIM. Since most designs contain many instances of a particular transistor configuration, only a small number of different configurations would need to be analyzed. Thus, developing functional models for switch-level networks poses a great challenge but should yield many applications.

Appendix I - Multi-Port Networks

This appendix contains derivations of several equations and proofs of some properties regarding passive linear resistor networks.

I.1 Introduction

Suppose a time-invariant network N contains only voltage sources and passive, linear resistors in which each voltage source has its negative terminal connected to the reference node GND and is set to a voltage $0.0 \leq v \leq V_{dd}$. Assume furthermore that each node is connected by some path to a voltage source and that voltage sources are never connected in parallel, and hence all node voltages are well-defined. The corresponding *zero-state* network N_0 is defined as the network formed when all voltage sources in N are set to 0.0, i.e. they are short-circuited. The network N_0 contains only passive, linear resistors.

A port in the network consists of two terminals with any node serving as the positive terminal,[1] and the reference node GND serving as the negative terminal. We will be interested in networks with at most three ports, labeled 1 through 3. The indices i and j are used to denote any of these ports. We require only information about N and N_0 which can be observed at the ports. Otherwise the networks can be considered "black boxes".

1. Nodes which are the positive terminals of voltage sources in N correspond to input nodes in the switch-level network, while other nodes correspond to normal nodes. Either kind of node can be a positive port terminal.

I.2 Port Parameters

The *open-circuit impedance* parameters are defined in terms of the voltages measured at the ports of the zero-state network N_0 when a current source is connected across one of the ports. That is, if current source I is connected across port i, and a voltage v'_j is measured at port j, then z_{ji} is defined as

$$z_{ji} = \frac{v'_j}{I}.$$

These parameters form a matrix **Z** which in our case has dimension 3×3. Most presentations of multi-port networks [15] describe the analogous 2×2 matrix for two-port networks. The diagonal elements of **Z** are the "self-impedance" terms, i.e. the impedance measured across each port of N_0 when all other ports are left open-circuited. The off-diagonal elements are the "cross-impedance" terms indicating the degree to which the positive terminals of the two ports are connected. That is, z_{ij} equals 0.0 if there is no path in N_0 between the positive terminals of ports i and j,[1] or if one of these two terminals is connected directly GND through a zero-state voltage source. At the other extreme, z_{ij} equals z_{jj} if every path in N_0 from the positive terminal of port i to GND passes through the positive terminal of port j.

For a passive resistor network, the matrix **Z** obeys several important properties. First all elements are nonnegative and finite. Second, since the network is reciprocal, the matrix is symmetric:

$$z_{ij} = z_{ji}, \text{ for all i and j.}$$

This follows directly from the Reciprocity Theorem [15]. Third, for $z_{ii} > 0.0$, when a voltage source with a positive voltage v is connected directly across port i of N_0, while all other ports are left open-circuited, it will produce the same effect as a current source with current v/z_{ii}. Therefore, the voltage at any port j

1. As the term "path" is used here, a path cannot contain the reference node GND or any node connected directly to a voltage source as an intermediate node.

under these conditions will be

$$v'_j = \frac{z_{ji}}{z_{ii}} v .$$

This voltage cannot be less than 0.0 or greater than v, and therefore

$$0.0 \leq z_{ji} \leq z_{ii}, \text{ for all i and j.} \qquad \text{(A1.1)}$$

This result also holds when $z_{ii} = 0.0$, because this case occurs only when the positive terminal of port i in N_0 is connected directly to GND by a zero-state voltage source, and hence $z_{ji} = 0.0$.

The *open-circuit* voltage parameters of N are defined as the voltages v_1, v_2, and v_3 measured at ports 1, 2, and 3 of N with all three ports open-circuited. Parameters of this nature have not been found in any other presentations on multi-port networks. In general, these voltages will not be completely independent of one another when the network contains paths between the positive port terminals. To derive some of their relations, suppose that a voltage source of voltage v_i is connected across port i of N. Then no voltages in N will be changed. Now if all voltage sources in N are set to 0.0, the voltage at any port j will be given by

$$v'_j = \frac{z_{ji}}{z_{ii}} v_i ,$$

when $z_{ii} \neq 0.0$. Furthermore, since this voltage was obtained by changing the settings on some voltage sources from nonnegative values to 0.0, it must be less than or equal to the original open-circuit voltage of port j in N, and therefore

$$0.0 \leq \frac{z_{ji}}{z_{ii}} v_i \leq v_j, \text{ for all i such that } z_{ii} \neq 0.0 \text{ and all j.} \qquad \text{(A1.2)}$$

The Thevenin equivalent of the network N at port i contains a voltage source set to the open-circuit port voltage and a resistor of conductance equal to the net conductance across the port with all sources set to 0.0. As long as the other ports are left open-circuited, the Thevenin equivalent of N at port i is described by the parameters:

$$v_{thev} = v_i$$
$$g_{thev} = 1.0/z_{ii}.$$

Suppose the resistor conductances are given by rational functions of ρ with degree greater than 0.0. Then the open-circuit parameters will also be given by rational functions of ρ with degree less than 0. Furthermore, since $0.0 \leq z_{ij}(\rho) \leq z_{ii}(\rho)$ for all values of ρ:

$$deg(z_{ij}) \leq deg(z_{ii}) < 0 \quad \text{for all i and j.}$$

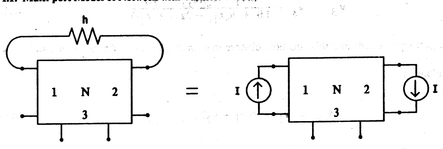
Similarly, the open-circuit voltage parameters will be given by rationpal functions of ρ, and since $0.0 \leq v_i(\rho) \leq V_{dd}$ for all values of ρ

$$deg(v_i) \leq 0 \quad \text{for all i.}$$

I.3 The Effect of a Variable Resistor

A network containing a single variable resistor can be described by a fixed network N with a variable resistor connected across ports 1 and 2, as shown in Figure I.1. Since the positive terminal of port 3 can be any node in the network, the port voltage can represent the voltage on any node in the network. Suppose the resistor has conductance h, and the resulting port voltages equal v'_1, v'_2, and v'_3.

Fig. I.1. Multi-port Model of Network with Variable Resistor



This resistor will have the effect of injecting a current I into port 1 of N and removing the current I from port 2, where

$$I = h(V'_1 - V'_2).$$

By superposition, the voltage at any port equals the sum of the voltage created by the sources of N, the voltage created by the current injected into port 1, and the voltage created by the current removed from port 2:

$$\begin{aligned} V'_1 &= V_1 + Iz_{11} - Iz_{21} \\ V'_i &= V_i + h(V'_1 - V'_2)(z_{i1} - z_{2i}) \end{aligned} \tag{A1.3}$$

For ports 1 and 2, this gives two equations in the two unknowns V'_1 and V'_2:

$$\begin{aligned} V'_1 &= V_1 + h(V'_1 - V'_2)(z_{11} - z_{12}) \\ V'_2 &= V_2 + h(V'_1 - V'_2)(z_{12} - z_{22}) \end{aligned}$$

This set of equations can be solved to give

$$V'_1 - V'_2 = \frac{V_1 - V_2}{1.0 + h(z_{11} - 2z_{12} + z_{22})}$$

and this result can be substituted into equation A1.3 to give

$$V'_i = V_i + \frac{h(V_1 - V_2)(z_{i1} - z_{2i})}{1.0 + h(z_{11} - 2z_{12} + z_{22})} \tag{A1.4}$$

In particular, the voltage at port 3 will be given by

$$V'_3 = V_3 + \frac{h(V_1 - V_2)(z_{13} - z_{23})}{1.0 + h(z_{11} - 2z_{12} + z_{22})}. \tag{A1.5}$$

To gain some insight into this equation, observe that for small values of h, $V'_1 - V'_2 \approx V_1 - V_2$, and therefore

$$V'_3 \approx V_3 + h(V_1 - V_2)(z_{13} - z_{23}). \tag{A1.6}$$

In other words, the denominator in equation A1.5 approximately equals 1.0. As h becomes very large, the

increased conductance between the two nodes will be offset by the decreased voltage difference, and hence each node voltage in the network approaches an asymptote:

$$\lim_{h \to \infty} v'_3 = v_3 + \frac{(v_1 - v_2)(z_{13} - z_{23})}{z_{11} - 2z_{12} + z_{22}}. \qquad \text{(A1.7)}$$

Thus v_i is approximately a linear function of h for small values of h and levels off to a constant value for large h.

The general form of equation A1.5 is

$$v(h) = v(0.0) + \frac{a\,h}{1.0 + b\,h}. \qquad \text{(3.4)}$$

Furthermore, since $z_{12} \leq z_{11}$ and $z_{12} \leq z_{22}$

$$b = z_{11} - 2z_{12} + z_{22} = (z_{11} - z_{12}) + (z_{22} - z_{12}) \geq 0.0. \qquad \text{(A1.8)}$$

Equations 3.4 and A1.8 are sufficient to prove Lemma 3.1.

We can derive further properties of equation A1.5. When a positive current I is injected into port 1 and removed from port 2 of N_0, the positive terminal of port 1 must have the maximum node voltage in the network and the positive terminal of port 2 must have the minimum. Therefore

$$I(z_{12} - z_{22}) \leq I(z_{13} - z_{23}) \leq I(z_{11} - z_{12})$$
$$(z_{12} - z_{22}) \leq (z_{13} - z_{23}) \leq (z_{11} - z_{12}).$$

The second inequality holds when I is negative as well. From this one can see that

$$|z_{13} - z_{23}| \leq z_{11} - 2z_{12} + z_{22}. \qquad \text{(A1.9)}$$

If the resistor conductances are given by rational functions of ρ, then the voltages will also be a rational function of ρ and will have a general form

$$v(\rho, h(\rho)) = v(\rho, 0.0) + \frac{a(\rho)\,h(\rho)}{1.0 + b(\rho)\,h(\rho)}.$$

Furthermore, since the impedance parameters all have degrees less than 0, $deg(b) < 0$, and since equation

A1.9 holds for all values of ρ:

$$deg((v_1 - v_2)(z_{13} - z_{23})) \leq deg(z_{13} - z_{23}) \leq deg(z_{11} - 2z_{12} + z_{22}).$$

Therefore

$$deg(a) \leq deg(b) < 0.$$

These results are sufficient to prove Lemma 3.3.

As a historical note, although equation A1.5 would seem to have other applications in electrical network theory, no such result has been found in the literature. Some presentations on sensitivity analysis (such as [15, p.678]) derive the approximate equation A1.6 for small values of h, but the more general result is not given. The technique of viewing the variable resistor as a current source of unknown current and then later solving simultaneous equations to find this current is generally credited to Kron [25] which he used in a method called "diakoptics". More recently, this technique has been applied to solving systems of sparse equations by a method called "tearing" [7, 32].

I.4 The Effect of Connecting Two Ports

We can determine the effects of connecting together ports 1 and 2 in network N by letting the conductance h in the previous development approach infinity. We will assume that $z_{11} > 0.0$ and $z_{22} > 0.0$, i.e. there is no voltage source connected directly across ports 1 or 2 of N.[1] In the limit the voltages on the two ports will approach a single voltage v where

$$v = \lim_{h \to \infty} v'_1 = v_1 + \frac{(v_1 - v_2)(z_{11} - z_{21})}{z_{11} - 2z_{12} + z_{22}}.$$

By rearranging terms we get

$$v = \frac{(z_{22} - z_{12})v_1 + (z_{11} - z_{12})v_2}{z_{11} - 2z_{12} + z_{22}}.$$

---

1. This restriction can be assumed, because the combination rule is never applied directly to the logic signals which describe input nodes.

If we define k_1 and k_2 as

$$k_1 = z_{12}/z_{11}$$
$$k_2 = z_{12}/z_{22}$$

then

$$v_{thev} = \frac{g_1(1.0-k_2)v_1 + g_2(1.0-k_1)v_2}{g_1(1.0-k_2) + g_2(1.0-k_1)} \qquad (4.4)$$

where g_1 and g_2 are the Thevenin conductances at ports 1 and 2, i.e. the reciprocals of z_{11} and z_{22}, respectively, and from our assumptions both values must be positive and finite. The following properties of the factors k_1 and k_2 can be derived from their definitions and from equations A1.1 and A1.2:

$$g_2k_1 = g_1k_2$$
$$0.0 \leq k_1 \leq 1.0$$
$$0.0 \leq k_2 \leq 1.0$$
$$k_1 \cdot v_1 \leq v_2$$
$$k_2 \cdot v_2 \leq v_1.$$

Equation 4.4 and the above properties of the factors k_1 and k_2 are sufficient to prove the validity of the combination rule for cyclic connections, as is done in Chapter 4.

Appendix II - Proofs of Results in Chapter 6

This appendix contains proofs of Theorem 6.4, Corollary 6.4.1, and Theorems 6.5 and 6.7..

II.1 Passive Networks

The method of conditioned relaxations relies on the fact that logical conductances act as passive elements. That is, when a signal is coupled through a logical conductance, the resulting signal has strength less than or equal to the original signal strength. Furthermore, signal combination always ignores the weaker signal. This allows us to kill any signal on a node with strength less than the steady state signal strength, knowing that no possible action of the network could amplify this signal into one critical to the formation of some steady state signal.

The passiveness of logical conductances has important implications on the recurrence equations describing a logical conductance network. Before the desired theorems can be proved, some general properties of "passive" recurrence equations must be derived. For a value $s \in \mathcal{T}$ let a_s denote the vector *block* (a, [s]), where [s] denotes a vector with each element equal to s. Similarly, for a function $f:\mathcal{T}^n \rightarrow \mathcal{T}^n$, let f_s denote the function $f_s(a) = block(f(a), [s])$.

A function $f:\mathcal{T}^n \rightarrow \mathcal{T}^n$ is said to be *passive* if for any s and any a, $f_s(a) = f_s(a_s)$. In other words, if $b = f(a)$ then any element of b greater than or equal to s can depend only on elements of a greater than or equal to s. No element of a less than s will be amplified into a value greater than or equal to s. A function of more than one argument is passive if it is passive for each argument. The functions $\uparrow$, $\downarrow$, $*$, *block* and constant functions are all passive, as is any composition of passive functions. The successor function *suc*, on the other hand, is not, where *suc* is defined as $suc(\gamma_p) = \gamma_{p'}$ and for any $a < \gamma_{p'}$, *suc*(a) equals the least element of $\mathcal{T}$ greater than a.

The following lemma shows the relation between the minimum solutions of the equations $\mathbf{a} = f(\mathbf{a})$ and $\mathbf{a} = f_s(\mathbf{a})$.

Lemma A2.1.
For a monotonic and passive function $f:\gamma^n \to \gamma^n$, if $\mathbf{a}^{min}$ is the minimum solution of the equation $\mathbf{a} = f(\mathbf{a})$ then $\mathbf{a}^{min}_s$ is the minimum solution of the equation $\mathbf{a} = f_s(\mathbf{a})$.

Proof of Lemma A2.1:

We will show by induction on k that

$$f^k(0)_s = f_s^k(0).$$

This clearly holds for $k = 0$, so assume it holds for k-1.

$$f^k(0)_s = f_s(f^{k-1}(0)) = f_s(f^{k-1}(0)_s) = f_s(f_s^{k-1}(0)) = f_s^k(0).$$

The minimum solution of $\mathbf{a} = f_s(\mathbf{a})$ is given by

$$\lim_{k \to \infty} f_s^k(0) = \lim_{k \to \infty} f^k(0)_s = \mathbf{a}^{min}_s. \qquad \blacksquare$$

Observe that this property may not hold if the function f is not passive. For example, the minimum solution of the equation $\mathbf{a} = suc(\mathbf{a})$ equals γ_p, but for any $s > 0$, the minimum solution of the equation $\mathbf{a} = suc_s(\mathbf{a})$ equals 0, even though the function *suc* is monotonic.

We can now prove a lemma that expresses in a very distilled form that the method of conditioned relaxations will not lead to a "runt" solution with some signals weaker than the steady state signals.

Lemma A2.2.

For a monotonic and passive function $f: \mathbf{y}^{[l]} \rightarrow \mathbf{y}^{[l]}$, if $\mathbf{a}^{min}$ is the minimum solution of the equation $\mathbf{a} = f(\mathbf{a})$ then $\mathbf{a}^{min}$ is also the minimum solution of the equation $\mathbf{a} = f'(\mathbf{a})$ where

$$f'(\mathbf{a}) = block(f(\mathbf{a}), \mathbf{a}^{min}).$$

Proof of Lemma A2.2:

Let $\mathbf{a}^{min'}$ equal the minimum solution of the equation $\mathbf{a} = f'(\mathbf{a})$. Clearly $\mathbf{a}^{min}$ is also a solution of this equation and therefore $\mathbf{a}^{min} \geq \mathbf{a}^{min'}$. We will prove that the two vectors are in fact equal by induction on decreasing strength values. By Lemma A2.1, $\mathbf{a}^{min}{}_s$ and $\mathbf{a}^{min'}{}_s$ are the minimum solutions of the equations $\mathbf{a} = f_s(\mathbf{a})$, and $\mathbf{a} = f'_s(\mathbf{a})$, respectively for any value of s. The function *block* obeys the following identities

$$block_{\gamma_p}(c, d) = c_{\gamma_p}$$
$$block_s(c, d) = block(c_s, d_{suc(s)}).$$

That is, unless the second argument is greater than the first, *block* behaves as an identity function of its first argument. Therefore

$$f'_{\gamma_p}(\mathbf{a}) = block_{\gamma_p}(f(\mathbf{a}), \mathbf{a}^{min}) = f_{\gamma_p}(\mathbf{a}).$$

This implies that $\mathbf{a}^{min}{}_{\gamma_p} = \mathbf{a}^{min'}{}_{\gamma_p}$. Now suppose $\mathbf{a}^{min}{}_{suc(s)} = \mathbf{a}^{min'}{}_{suc(s)}$. We will show that $\mathbf{a}^{min}{}_s = \mathbf{a}^{min'}{}_s$.

$$\mathbf{a}^{min'}{}_s = f'_s(\mathbf{a}^{min'}{}_s) = block(f_s(\mathbf{a}^{min'}{}_s), \mathbf{a}^{min}{}_{suc(s)}) = block(f_s(\mathbf{a}^{min'}{}_s), \mathbf{a}^{min'}{}_{suc(s)}).$$

We know that $\mathbf{a}^{min'}{}_{suc(s)} \leq \mathbf{a}^{min'}{}_s$, and therefore the left hand argument of *block* in the above equation must be greater than or equal to the right hand argument. This means that this application of *block* will behave as an identity function of its first argument, giving

$$\mathbf{a}^{min'}{}_s = f_s(\mathbf{a}^{min'}{}_s).$$

which shows that $a^{out'}_S = a^{out}_S$. The set J is totally ordered and finite. Hence by induction on decreasing strength values $a^{out'} = a^{out}$. ∎

This result also may not hold for a function J which is not passive. For example, the equation $a = suc(a)$ has a minimum solution γ_P but the equation $a = block\ (suc(a), \gamma_P)$ has a minimum solution 0.

The result of Lemma A2.2 can be expressed in a form closer to what is required to prove Theorem 6.4.

Lemma A2.3.
For a matrix $G \in \mathcal{Y}^{m \times n}$, and a vector $\phi \in \mathcal{U}^n$, define the functions f_1 and f_0 as:

$$f_1(a) = block\ (f \phi \uparrow G \cdot a, r)$$
$$f_0(d) = block\ (\lfloor d \rfloor \uparrow G \cdot \delta, r).$$

where

$$r = G^{\circ} \cdot \lfloor \phi \rfloor.$$

If a^{out} and d^{out} are the minimum solutions of the equations $a = f_1(a)$ and $d = f_0(d)$, respectively, then

$$r = a^{out} \uparrow d^{out}$$

Proof of Lemma A2.3:

Define the function g as

$$g(a) = block\ (\lfloor \phi \rfloor \uparrow G \cdot a, r).$$

Lemma A2.2, combined with Corollary 6.2.1 shows that r must be the minimum solution of the equation $a = g(a)$. We will show by induction on k that

$$g^k(0) = f_1^k(0) \uparrow f_0^k(0).$$

Clearly this holds for k = 0. Now suppose that it holds for k-1.

$$g^k(0) = block(\|b\| \uparrow G \cdot g^{k-1}(0), r) = block(\|b\| \uparrow G \cdot (f_1^{k-1}(0) \uparrow f_0^{k-1}(0)), r)$$
$$g^k(0) = block(\lceil b \rceil \uparrow G \cdot f_1^{k-1}(0), r) \uparrow block(\lfloor b \rfloor \uparrow G \cdot f_0^{k-1}(0), r) = f_1^k(0) \uparrow f_0^k(0).$$

Taking the limit of this equation gives the desired result.

$$r = \lim_{k \to \infty} g^k(0) = \lim_{k \to \infty} f_1^k(0) \uparrow \lim_{k \to \infty} f_0^k(0) = u^{min} \uparrow d^{min}. \quad \blacksquare$$

II.2 Theorem 6.4

Lemma A2.3 takes care of the most difficult part of the proof of Theorem 6.4.

Theorem 6.4.
For a matrix $G \in \mathcal{Y}^{n \times n}$, and a vector $b \in \mathcal{A}^n$, define G as $G = xG$. The unique minimum solution of the equation $a = f(a)$, where

$$f(a) = b \vee G \circ a \tag{6.19}$$

is given by

$$a^{min} = +u^{min} \vee -d^{min},$$

where u^{min} and d^{min} are the minimum solutions of the equations $u = f_1(u)$ and $d = f_0(d)$, respectively, and the functions f_1 and f_0 are defined as:

$$f_1(u) = block(\lceil b \rceil \uparrow G \cdot u, r) \tag{6.20}$$
$$f_0(d) = block(\lfloor b \rfloor \uparrow G \cdot d, r), \tag{6.21}$$

and

$$r = G^* \cdot \|b\|. \tag{6.22}$$

Proof of Theorem 6.4:

Define the vector $\mathbf{k}$ as $\mathbf{k} = +\mathbf{u}^{\infty} \vee -\mathbf{d}^{\infty}$. We will first show that $\mathbf{k}$ satisfies the recurrence relation $\mathbf{k} = \mathcal{J}(\mathbf{k})$. Observe that $\Gamma\mathbf{A}\uparrow = \mathbf{u}^{\infty}$ and $\mathsf{L}\mathbf{A}\downarrow = \mathbf{d}^{\infty}$. Lemma A2.3 then shows that

$$\mathbf{r} = \mathbf{u}^{\infty} \uparrow \mathbf{d}^{\infty} = \|\mathbf{A}\| = \|\mathbf{b}\| \uparrow \mathbf{G} \cdot \|\mathbf{A}\| = \|\mathbf{b} \vee \mathbf{G} \circ \mathbf{A}\|.$$

Substituting this into the definition of $\mathcal{J}_1$ gives

$$\mathcal{J}_1(\Gamma\mathbf{A}\uparrow) = \mathit{Block}(\Gamma\mathbf{b}\uparrow \mathbf{G} \cdot \Gamma\mathbf{A}\uparrow, \|\mathbf{b} \vee \mathbf{G} \circ \mathbf{A}\|) = \Gamma\mathbf{b} \vee \mathbf{G} \circ \mathbf{A}\uparrow.$$

Similarly,

$$\mathcal{J}_0(\mathsf{L}\mathbf{A}\downarrow) = \mathsf{L}\mathbf{b} \vee \mathbf{G} \circ \mathbf{A}\downarrow.$$

Since $\Gamma\mathbf{A}\uparrow = \mathbf{u}^{\infty}$, $\Gamma\mathbf{A}\uparrow = \mathcal{J}_1(\Gamma\mathbf{A}\uparrow)$, and similarly $\mathsf{L}\mathbf{A}\downarrow = \mathcal{J}_0(\mathsf{L}\mathbf{A}\downarrow)$. This shows that

$$\mathbf{A} = +\Gamma\mathbf{A}\uparrow \vee -\mathsf{L}\mathbf{A}\downarrow = +\Gamma\mathbf{b} \vee \mathbf{G} \circ \Gamma\mathbf{A}\uparrow \vee -\mathsf{L}\mathbf{b} \vee \mathbf{G} \circ \mathsf{L}\mathbf{A}\downarrow = \mathbf{b} \vee \mathbf{G} \circ \mathbf{A}.$$

Therefore $\mathbf{A}$ satisfies the recurrence relation $\mathbf{A} = \mathcal{J}(\mathbf{A})$, which implies that $\mathbf{A} \geq \mathbf{d}^{\infty}$. By the monotonicity of $\|\ \|$, Γ, $\uparrow$, and $\mathsf{L}\downarrow$:

$$\|\mathbf{A}\| \geq \|\mathbf{d}^{\infty}\| \tag{A2.10}$$

$$\mathbf{u}^{\infty} = \Gamma\mathbf{A}\uparrow \geq \Gamma\mathbf{d}^{\infty}\uparrow \tag{A2.11}$$

$$\mathbf{d}^{\infty} = \mathsf{L}\mathbf{A}\downarrow \geq \mathsf{L}\mathbf{d}^{\infty}\downarrow. \tag{A2.12}$$

We can now show that $\mathbf{A}$ must equal $\mathbf{d}^{\infty}$ by factoring the equation $\mathbf{d}^{\infty} = \mathcal{J}(\mathbf{d}^{\infty})$. First we can find the strength of $\mathbf{d}^{\infty}$ as

$$\|\mathbf{d}^{\infty}\| = \|\mathbf{b} \vee \mathbf{G} \circ \mathbf{d}^{\infty}\| = \|\mathbf{b}\| \uparrow \mathbf{G} \cdot \|\mathbf{d}^{\infty}\|.$$

Therefore $\|\mathbf{d}^{\infty}\|$ must be greater than or equal to the minimum solution of this recurrence relation, i.e. $\|\mathbf{d}^{\infty}\| \geq \mathbf{r} = \|\mathbf{A}\|$. Combining this result with the inequality of equation A2.10 shows that $\|\mathbf{d}^{\infty}\| = \|\mathbf{A}\|$. Now we can see that

$$\Gamma\mathbf{d}^{\infty}\uparrow = \Gamma\mathbf{b} \vee \mathbf{G} \circ \mathbf{d}^{\infty}\uparrow = \mathit{Block}(\Gamma\mathbf{b}\uparrow \mathbf{G} \cdot \Gamma\mathbf{d}^{\infty}\uparrow, \|\mathbf{d}^{\infty}\|)$$

$$\Gamma a^{min}\urcorner = block(\Gamma b\urcorner \uparrow G \bullet \Gamma a^{min}\urcorner, r) = f_1(\Gamma a^{min}\urcorner).$$

Therefore $\Gamma a^{min}\urcorner \geq u^{min}$, which when combined with the inequality of equation A2.11 gives $\Gamma a^{min}\urcorner = u^{min}$. A similar derivation gives $\llcorner a^{min}\lrcorner = d^{min}$. We can now complete the proof with

$$a^{min} = +\Gamma a^{min}\urcorner \vee -\llcorner a^{min}\lrcorner = +u^{min} \vee -d^{min}. \qquad \blacksquare$$

II.3 Corollary 6.4.1

Corollary 6.4.1.
For a matrix $G \in \mathcal{Y}^{n \times n}$, and a vector $b \in \mathcal{A}^n$, if G is defined as $G = \alpha(G)$, then the minimum solution of the equation $a = f(a)$ where

$$f(a) = b \vee G \circ a$$

is given by

$$a^{min} = \lim_{k \to \infty} f^k(0) \tag{6.24}$$

where

$$f'(a) = kill(f(a), r), \tag{6.25}$$

and

$$r = G^* \bullet \| b \|.$$

Proof of Corollary 6.4.1:

Expanding the equation for f' gives

$$f'(a) = kill(b \vee G \circ a, r) = +block(\Gamma b \vee G \circ a\urcorner, r) \vee -block(\llcorner b \vee G \circ a\lrcorner, r).$$

In the proof of Theorem 6.4 it is shown that $r = \| a^{min} \|$ and therefore for any $a \leq a^{min}$

$$r = \| a^{min} \| = \| b \| \uparrow G \bullet \| a^{min} \| \geq \| b \| \uparrow G \bullet \| a \| = \| b \vee G \circ a \|.$$

For any $a \leq a^{min}$

$$block(\Gamma b \vee G \circ a\urcorner, r) = block(block(\Gamma b\urcorner \uparrow G \bullet \Gamma a\urcorner, \| b \vee G \circ a \|), r)$$
$$block(\Gamma b \vee G \circ a\urcorner, r) = block(\Gamma b\urcorner \uparrow G \bullet \Gamma a\urcorner, r) = f_1(\Gamma a\urcorner),$$

where f_1 is defined in equation 6.20, and a similar result holds for the 0 part with respect to the function f_0 defined in equation 6.21. Then for $a \leq d^{\infty}$

$$f'(a) = +/_1(\Gamma a\urcorner) \vee -/_0(\llcorner a\lrcorner). \quad (A2.13)$$

We will show by induction on k that

$$f'^k(\emptyset) = +/_1^k(0) \vee -/_0^k(0) \quad (A2.14)$$

This clearly holds for k = 0, and assuming it holds for k implies that

$$\Gamma f'^k(\emptyset)\urcorner = f_1^k(0)$$
$$\llcorner f'^k(\emptyset)\lrcorner = f_0^k(0)$$

For u^{∞} and d^{∞} defined in the statement of Theorem 6.4, $f_1^k(0) \leq u^{\infty}$ and $f_0^k(0) \leq d^{\infty}$. Therefore, by the monotonicity of +/, -/, and $\vee$:

$$f'^k(\emptyset) = +/_1^k(0) \vee -/_0^k(0) \leq +u^{\infty} \vee -d^{\infty} = d^{\infty},$$

and equation A2.13 applies when $a = f'^k(\emptyset)$. This allows to expand f'^{k+1} as:

$$f'^{k+1}(\emptyset) = f'(f'^k(\emptyset)) = +/_1(\Gamma f'^k(\emptyset)\urcorner) \vee -/_0(\llcorner f'^k(\emptyset)\lrcorner)$$
$$f'^{k+1}(\emptyset) = +/_1(f_1^k(0)) \vee -/_0(f_0^k(0)) = +/_1^{k+1}(0) \vee -/_0^{k+1}(0).$$

which proves the induction assertion. Combining the result of Theorem 6.4 with equation A2.14 gives

$$d^{\infty} = +\lim_{k \to \infty} /_1^k(0) \vee -\lim_{k \to \infty} /_0^k(0) = \lim_{k \to \infty} (+/_1^k(0) \vee -/_0^k(0)).$$

and therefore

$$d^{\infty} = \lim_{k \to \infty} f'^k(\emptyset). \quad \blacksquare$$

II.4 Theorem 6.5

Theorem 6.5.

The target state $\tilde{y}$ of a switch-level network is given by

$$\tilde{y} = \langle +u^{opt} \rangle \sqcup \langle -d^{opt} \rangle, \tag{6.28}$$

where u^{opt} and d^{opt} are the minimum solutions of the equations $u = g_1(u)$ and $d = g_0(d)$, respectively, and the functions g_1 and g_0 are defined as

$$g_1(u) = block(E^{max} \bullet \lceil x \rceil \uparrow \lceil y \rceil \uparrow G^{max} \bullet u, r), \tag{6.29}$$

$$g_0(d) = block(E^{max} \bullet \lfloor x \rfloor \uparrow \lfloor y \rfloor \uparrow G^{max} \bullet d, r), \tag{6.30}$$

and

$$r = G^{max^*} \bullet (E^{max} \bullet \| x \| \uparrow \| y \|). \tag{6.31}$$

Proof of Theorem 6.5:

First, the idea behind the optimization method can be derived formally by algebraic manipulation.

$$\tilde{y} = \bigsqcup_{\{G\},\{E\}} \bar{y}(G, E) = \bigsqcup_{\{G\},\{E\}} \langle \pi(G, E) \rangle = \bigsqcup_{\{G\},\{E\}} \langle +\lceil \pi(G, E) \rceil \vee -\lfloor \pi(G, E) \rfloor \rangle$$

$$\tilde{y} = \bigsqcup_{\{G\},\{E\}} \langle +\lceil \pi(G, E) \rceil \rangle \sqcup \bigsqcup_{\{G\},\{E\}} \langle -\lfloor \pi(G, E) \rfloor \rangle.$$

The last step above follows from the identity shown in equation 5.9. It relies on the fact that unless the state of a signal equals X, either its 0 part or its 1 part must equal 0. Furthermore observe that the function of b whose value is $\langle +b \rangle$ is monotonic, and therefore

$$\langle +a \rangle \sqcup \langle +b \rangle = \langle +(a \uparrow b) \rangle$$

and that a similar identity holds with -. Hence

$$\tilde{y} = \langle +(\uparrow_{\{G\},\{E\}} \lceil \pi(G, E) \rceil) \rangle \sqcup \langle -(\uparrow_{\{G\},\{E\}} \lfloor \pi(G, E) \rfloor) \rangle. \tag{A2.15}$$

Since $\uparrow$ denotes the pointwise maximum operation, this equation shows that the target state for any node η_1 can be computed by finding the maximum values of $\Gamma \gamma_1(G, E)\urcorner$ and $\llcorner \gamma_1(G, E)\lrcorner$ for all $G \in \{ G \}$ and $E \in \{ E \}$.

Define $u^{min}(G, E)$ and $d^{min}(G, E)$ as the minimum solutions of the equation $u = f_1(u)$ and $d = f_0(d)$, respectively, where

$$f_1(u) = block(E \cdot \Gamma x \urcorner \uparrow \Gamma y \urcorner \uparrow G \cdot u,\ G^*(E \cdot \| x \| \uparrow \| y \|)).$$
$$f_0(d) = block(E \cdot \llcorner x \lrcorner \uparrow \llcorner y \lrcorner \uparrow G \cdot d,\ G^*(E \cdot \| x \| \uparrow \| y \|)).$$

Theorem 6.4 shows that

$$u^{min}(G, E) = \Gamma \kappa(G, E)\urcorner$$
$$d^{min}(G, E) = \llcorner \kappa(G, E) \lrcorner.$$

Therefore, if we can prove that

$$u^{opt} = \uparrow_{\{G\}, \{E\}} u^{min}(G, E) \tag{A2.16}$$
$$d^{opt} = \uparrow_{\{G\}, \{E\}} d^{min}(G, E). \tag{A2.17}$$

then comparing equation 6.28 to equation A2.15, it can be seen that the theorem will be proved.

Only the proof of equation A2.16 will be shown. Equation A2.17 follows identically. For any $G \in \{ G \}$, $E \in \{ E \}$ and $u \in \gamma^n$

$$E \cdot \Gamma x \urcorner \uparrow \Gamma y \urcorner \uparrow G \cdot u \leq E^{max} \cdot \Gamma x \urcorner \uparrow \Gamma y \urcorner \uparrow G^{max} \cdot u$$
$$G^* \cdot (E \cdot \| x \| \uparrow \| y \|) \geq G^{max^*} \cdot (E^{min} \cdot \| x \| \uparrow \| y \|).$$

Therefore, since *block* is monotonic in its first argument and antimonotonic in its second, $f_1 \leq g_1$. Applying Theorem 6.3 gives $u^{min}(G, E) \leq u^{opt}$ for any G and E in the allowed range. Therefore

$$\uparrow_{\{G\}, \{E\}} u^{min}(G, E) \leq u^{opt}.$$

To complete the proof, we need only find some setting of G and E which gives $u^{min}(G, E) = u^{opt}$.

Define the matrices G′ and E′ as follows:

$$g'_{ij} = \begin{cases} g^{min}_{ij}, & u^{opt}_i = 0 \text{ or } u^{opt}_j = 0 \\ g^{max}_{ij}, & u^{opt}_i > 0 \text{ and } u^{opt}_j > 0, \end{cases}$$

$$e'_{ij} = \begin{cases} e^{min}_{ij}, & \lceil x_j \rceil = 0 \\ e^{max}_{ij}, & \lceil x_j \rceil > 0. \end{cases}$$

We will show that $u^{min}(G', E') = u^{opt}$. From these definitions, we can see that

$$E' \cdot \lceil x \rceil = E^{max} \cdot \lceil x \rceil, \tag{A2.18}$$

and can show that for any $u \leq u^{opt}$

$$block\,(G' \cdot u, r) = block\,(G^{max} \cdot u, r). \tag{A2.19}$$

To see this, observe that for $u \leq u^{opt}$,

$$block\,(G' \cdot u, r) \leq block\,(G^{max} \cdot u, r) \leq block\,(G^{max} \cdot u^{opt}, r) \leq block\,(u^{opt}, r).$$

Therefore, for $u^{opt}_i = 0$,

$$0 \leq block\,([G' \cdot u]_i, r_i) \leq block\,(u^{opt}_i, r_i) = 0,$$

and for $u^{opt}_i > 0$,

$$block\,([G' \cdot u]_i, r_i) = block\,(\underset{u^{opt}_j > 0}{\uparrow}(g'_{ij} \downarrow u_j), r_i)$$
$$block\,([G' \cdot u]_i, r_i) = block\,(\underset{u^{opt}_j > 0}{\uparrow}(g^{max}_{ij} \downarrow u_j), r_i)$$
$$block\,([G' \cdot u]_i, r_i) = block\,([G^{max} \cdot u]_i, r_i).$$

Equations A2.18 and A2.19 imply that u^{opt} is a solution to the equation

$$u = block\,(E' \cdot \lceil x \rceil \uparrow \lceil y \rceil \uparrow G' \cdot u, r) \tag{A2.20}$$

and furthermore any solution less than or equal to u^{opt} must also satisfy the equation $u = g_1(u)$, and therefore u^{opt} is the minimum solution of equation A2.20. Lemma A2.2 then shows that u^{opt} must also be

the minimum solution of the equation

$$u = block(E' \cdot \lceil x \rceil \uparrow \lceil y \rceil \uparrow G' \cdot u,\ r \uparrow u^{opt}).$$

Let $s = G'^{*} \cdot (E' \cdot \| x \| \uparrow \| y \|)$. It is claimed that $s = r \uparrow u^{opt}$. Clearly

$$r = G^{min^{*}} \cdot (E^{min} \cdot \| x \| \uparrow \| y \|) \leq G'^{*} \cdot (E' \cdot \| x \| \uparrow \| y \|) = s.$$

We know that s is the minimum solution of the equation $a = h(a)$, where

$$h(a) = E' \cdot \| x \| \uparrow \| y \| \uparrow G' \cdot a.$$

This shows that $s \geq u^{opt}$, because the function in the recurrence equation A2.20 is less than or equal to h. Therefore $s \geq r \uparrow u^{opt}$. Next we will show that $r \uparrow u^{opt} = h(r \uparrow u^{opt})$, which will prove that $r \uparrow u^{opt} \geq s$, and when combined with the previous inequality, shows that $r \uparrow u^{opt} = s$. First observe that

$$G' \cdot (r \uparrow u^{opt}) = G^{min} \cdot r \uparrow G' \cdot u^{opt}.$$

This follows because if $r_i \geq u^{opt}_i$, then $u^{opt}_i = 0$, which implies that the ith column of G' will equal the ith column of G^{min}. Similarly

$$E' \cdot \| x \| = E^{min} \cdot \| x \| \uparrow E' \cdot \lceil x \rceil.$$

We also know that r satisfies the recurrence relation

$$r = E^{min} \cdot \| x \| \uparrow \| y \| \uparrow G^{min} \cdot r.$$

Combining these facts with equation A2.20 gives

$$\begin{aligned}
u^{opt} \uparrow r &= E' \cdot \lceil x \rceil \uparrow \lceil y \rceil \uparrow G' \cdot u^{opt} \uparrow r \\
u^{opt} \uparrow r &= E' \cdot \lceil x \rceil \uparrow E^{min} \cdot \| x \| \uparrow \lceil y \rceil \uparrow \| y \| \uparrow G' \cdot u^{opt} \uparrow G^{min} \cdot r \\
u^{opt} \uparrow r &= E' \cdot \| x \| \uparrow \| y \| \uparrow G' \cdot (u^{opt} \uparrow r) \\
u^{opt} \uparrow r &= h(r \uparrow u^{opt}).
\end{aligned}$$

Putting these results together, we have shown that u^{opt} is the minimum solution of the equation

$$u = block\,(E^{\prime}\cdot\Gamma x\uparrow\uparrow\Gamma y\uparrow\uparrow G^{\prime}*u,\ G^{0}*(E^{\prime}\cdot\|x\|\uparrow\uparrow\|y\|)).$$

and therefore $u^{opt} = \Gamma x(G^{\prime}, E^{\prime})\uparrow\uparrow\Gamma y(G^{\prime}, E^{\prime})\uparrow\uparrow u^{opt}(G; E)$. This completes our proof that

$$\mathbf{u^{opt}} = \uparrow_{\{G\},\{E\}} \mathbf{u^{opt}(G; E)}.$$

A similar result can be proved for $\mathbf{d^{opt}}$, which completes the proof of the theorem.∎

II.5 Theorem 6.7.

Theorem 6.7.
If $\tilde{y} = target\,(x, y, z)$, then $\tilde{y} = target\,(x, \tilde{y}, z)$.

Proof of Theorem 6.7:

Define the vector $\mathbf{y}$ as the set of signals formed by the normal modes in state $\tilde{y}$, i.e.

$$\mathbf{<\tilde{y}>} = \mathbf{\tilde{y}}$$
$$\mathbf{\|\tilde{y}\|} = \mathbf{cap}.$$

Observe that $\|y\| = \|\tilde{y}\| = cap$, and therefore for r defined in equation 6.7I

$$r = C^{cap}*(E^{cap}\cdot\|x\|\uparrow\|y\|) = C^{cap}*(E^{cap}\cdot\|x\|\uparrow\|\tilde{y}\|).$$

Compare the functions

$$g_1(u) = block\,(E^{cap}\cdot\Gamma x\uparrow\uparrow\Gamma y\uparrow\uparrow C^{cap}*u, x)$$
$$\tilde{g}_1(u) = block\,(E^{cap}\cdot\Gamma x\uparrow\uparrow\Gamma\tilde{y}\uparrow\uparrow C^{cap}*u, x)$$

Let u^{opt} and $\tilde{u}^{opt}$ be the minimum solutions of the equations $u = g_1(u)$, and $u = \tilde{g}_1(u)$, respectively. We will show that these two vectors are equal. First, $\Gamma\tilde{y}\uparrow 1 = u^{opt}_1 cap_1$ for all i. Furthermore, $u^{opt}_i \geq block\,(\Gamma y\uparrow 1, i)$, and $\Gamma y\uparrow 1 \leq cap_i$ for all i, which implies that $\Gamma\tilde{y}\uparrow 1 \geq block\,(\Gamma y\uparrow 1, i)$. Therefore,

since *block* is monotonic in its first argument

$$block\ (\Gamma \tilde{\gamma} 1,\ t) \ \geq\ block\ (block\ (\Gamma \gamma 1,\ t),\ t) \ =\ block\ (\Gamma \gamma 1,\ t),$$

which shows that $\tilde{g}_1 \geq g_1$, and by Theorem 6.3, $\tilde{u}^{opt} \geq u^{opt}$. We will now show that unless two values are in fact equal by showing $u^{opt} = \tilde{g}_1(u^{opt})$. Since $u^{opt} \geq \Gamma \tilde{\gamma} 1 \geq block\ (\Gamma \tilde{\gamma} 1,\ t)$,

$$g_1(u^{opt}) \ =\ u^{opt} \ =\ u^{opt} \uparrow block\ (\Gamma \tilde{\gamma} 1,\ t) \ =\ \tilde{g}_1(u^{opt}) \uparrow block\ (\Gamma \gamma 1,\ t),$$

and furthermore $\tilde{g}_1(u^{opt}) \geq g_1(u^{opt}) = u^{opt} \geq block\ (\Gamma \gamma 1,\ t)$ which gives

$$u^{opt} \ =\ \tilde{g}_1(u^{opt}).$$

Thus $u^{opt} = \tilde{u}^{opt}$, and by a similar argument we can show that $d^{opt} = \tilde{d}^{opt}$, where $\tilde{d}^{opt}$ is the minimum solution of the equation

$$d \ =\ block\ (\Sigma^{max} \cdot \lfloor x \rfloor \uparrow \ \lfloor \tilde{\gamma} \rfloor \uparrow \Gamma^{max} \cdot d,\ t).$$

Theorem 6.5 shows that $target\ (x,\ \tilde{\gamma},\ z)$ is given by

$$target\ (x,\ \tilde{\gamma},\ z) \ =\ \langle +\tilde{u}^{opt} \rangle \sqcup \langle -\tilde{d}^{opt} \rangle \ =\ \langle +u^{opt} \rangle \sqcup \langle -d^{opt} \rangle \ =\ target\ (x,\ \gamma,\ z). \qquad \blacksquare$$

References

[1] Aho, A. V., J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Addison-Wesley (1974).

[2] Akino, T., *et al*, "Circuit Simulation and Timing Verification Based on MOS/LSI Mask Information", *Proceedings of the Sixteenth Design Automation Conference*, IEEE, New York (1979), 88-94.

[3] Alia, G., P. Ciompi, E. Martinelli, "LSI Components Modelling in a Three-Valued Functional Simulation", *Proceedings of the Fifteenth Design Automation Conference*, IEEE, New York (1978), 428.

[4] Baker, C. M., *Artwork Analysis Tools for VLSI Circuits*, M.S. thesis, MIT Department of EECS (June, 1980).

[5] Baker, C. M., and C. Terman, "Tools for Verifying Integrated Circuit Designs", *Lambda Magazine* (Fourth Quarter, 1980) 22-30.

[6] Bose, A. K., and S. A. Szygenda, "Detection of Static and Dynamic Hazards in Logic Nets", *Proceedings of the Fourteenth Design Automation Conference*, IEEE, New York (1977), 216.

[7] Braae, R., *Matrix Algebra for Electrical Engineers*, London, Sir Isaac Pitman and Sons (1963).

[8] Bryant, R. E., *MOSSIM: A Logic-Level Simulator for MOS LSI, User's Manual*, Integrated Circuit Memo 80-21, MIT Department of EECS (July, 1980).

[9] Bryant, R. E., "An Algorithm for MOS Logic Simulation", *Lambda Magazine* (Fourth Quarter, 1980) 46-53.

[10] Brzozowski, J. A., and M. Yoeli, *Digital Networks*, Prentice-Hall (1976).

[11] Brzozowski, J. A., and M. Yoeli, "On a Ternary Model of Gate Networks", *IEEE Transactions on Computers*, vol. C-28, no. 3 (March, 1979), 178-183.

[12] Case, G. R., and J. D. Stauffer, "SALOGS-IV: A Program to Perform Logic Simulation and Fault Diagnosis", *Proceedings of the Fifteenth Design Automation Conference*, IEEE, New York (1978), 392-397.

[13] Chawla, B. R., H. K. Gummel, and P. Kozak, "MOTIS - An MOS Timing Simulator", *IEEE Transactions on Circuits and Systems*, IEEE, New York, vol. CAS-22, no. 12 (December, 1975), 901-909.

[14] Dennis, J. B., and S. Patil, "Speed Independent Aynchronous Circuits", *Fourth Hawaii International Conference on System Sciences*, Honolulu, Hawaii (January, 1971).

[15] Desoer, C. A., and E. S. Kuh, *Basic Circuit Theory*, New York, McGraw-Hill (1969).

[16] El-Ziq, Y. M., and S. Y. H. Su, "Logic Design Automation of Diagnosable MOS Combinational Logic Networks", *Proceedings of the Fourteenth Design Automation Conference*, IEEE, New York (1977), 205-215.

[17] Garey, M. R., and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Co. (1979).

[18] Glasser, L. A., *The Analog Behavior of Digital Integrated Circuits*, VLSI Memo 80-36, MIT Dept. of EECS (December, 1980).

[19] Hartmann, R. F., "Design and Market Potential for Gate Arrays" *Lambda Magazine* (Fourth Quarter, 1980) 55-59.

[20] Hill, F. J., and G. R. Peterson, *Introduction to Switching Theory and Logic Design*, Wiley, New York (1974).

[21] Holloway, J., *et al*, *The SCHEME-79 Chip*, AI Memo 559, MIT Artificial Intelligence Laboratory (December, 1979).

[22] Huffman, D. A., "The Synthesis of Sequential Switching Circuits", *Journal of the Franklin Institute*, Vol. 257, Nos. 3-4 (March and April 1954) 161-190, 275-303.

[23] Jephson, J. S., R. P. McQuarrie, and R. E. Vogelsberg, "A Three-Level Design Verification System", *IBM Systems Journal*, vol. 8, no. 3 (1969), 178-188.

[24] Johannsen, D., "Bristle Blocks: A Silicon Compiler", *Proceedings of the Sixteenth Design Automation Conference*, IEEE, New York (1979), 310-313.

[25] Kron, G., *Diakoptics*, Macdonald (1963).

[26] Liskov, B. E., *et al*, *CLU Reference Manual*, MIT Laboratory for Computer Science TR-225, Cambridge, MA. (October 1979).

[27] Maclane, S., and G. Birkhoff, *Algebra*, MacMillan (1968).

[28] Mead, C., and L. Conway, *Introduction to VLSI Systems*, Addison Wesley, Reading, Mass. (1979).

[29] Muller, D. E., and W. S. Bartky, "A Theory of Asynchronous Circuits", *Proc. of an International Symposium on the Theory of Switching The Annals of the Computation Laboratory of Harvard University*, Part I. Harvard University Press, Cambridge, Mass. (1959), 204-243.

[30] Nagel, L., *SPICE2: A Computer Program to Simulate Semiconductor Circuits*, Technical Report UCB ERL-M250, Electronics Research Laboratory, University of California, Berkeley (May, 1975).

[31] Newton, A. R., *The Simulation of Large-Scale Integrated Circuits*, PhD Thesis, University of California, Berkeley, *available as* Technical Report UCB ERL M78/52, Electronics Research Laboratory, University of California, Berkeley (July, 1978).

[32] Noble, B., *Applied Linear Algebra*, Prentice-Hall (1969).

[33] Persky, G., D. N. Deutsch, and D. G. Schweikert, "LTX - A Minicomputer Based System for Automated LSI Layout," *Journal of Design Automation and Fault Tolerant Computing*, Vol. 1, No. 4 (1977), 217-255.

[34] Rowson, J. and J. Wipfli, *A MOS Logic Circuit Simulator*, SSP Memo #2200, California Institute of Technology (November 1978).

[35] Scott, D. S., "Continuous Lattices", *Toposes, Algebraic Logic and Logic*, (F. W. Lawvere, ed.), Springer-Verlag, Berlin (1972). Also available as Technical Monograph PRG-7, Oxford University.

[36] Seitz, C. L., "Self-Timed VLSI Systems", *Proceedings of the Caltech Conference on VLSI*, Pasadena, Ca. (January, 1979).

[37] Seitz, C. L., "System Timing", Chapter 7 in C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison Wesley, Reading, Mass. (1979).

[38] Shannon, C. E., *A Symbolic Analysis of Relay and Switching Circuits*, M.S. Thesis, MIT Department of Electrical Engineering (1937).

[39] Shannon, C. E., "A Symbolic Analysis of Relay and Switching Circuits", *Transactions of the AIEE*, Vol. 57 (1938) 713-723.

[40] Stoy, J. E., *Denotational Semantics: The Scott-Strachey Approach to Programming Language Theory*, MIT Press (1977).

[41] Sugarman, R. "VLSI Computing: a Tough Nut to Crack", *Spectrum*, IEEE, vol. 17, no. 1 (January 1980), 34-35.

[42] Tanabe, N., Nakamura, H., and K. Kawakita, "MOSTAP: An MOS Circuit Simulator for LSI Circuits", *Proceedings of the International Symposium on Circuits and Systems*, IEEE, Houston, Texas, (April, 1980).

[43] Unger, S. H., *Asynchronous Sequential Switching Circuits*, Wiley (1969).

[44] Utah, University of, VLSI Research Group, *Simulog Manual*, (April, 1979).

[45] Wilcox, P., "Digital Logic Simulation at the Gate and Functional Level", *Proceedings of the Sixteenth Design Automation Conference*, IEEE, New York (1979), 242-248.

[46] Yoeli, M., and S. Rinon, "Application of Ternary Algebra to the Study of Static Hazards," *Journal of the ACM*, ACM, New York, vol. 11, no. 1 (Jan., 1964), 84-97.

Biographical Note

Randal E. Bryant was born on October 27, 1952 in Mountainside, New Jersey. He grew up in Birmingham, Michigan and graduated from Seaholm High School in 1970. He received his B.S. degree in Applied Mathematics from the University of Michigan in December, 1973. Since September, 1974 he has attended graduate school in the Department of Electrical Engineering and Computer Science at MIT, receiving the S.M. degree in June, 1977 and the E.E. degree in February, 1978. During this time he has been associated with the Computation Structures Group in the Laboratory for Computer Science, headed by Professor Jack B. Dennis. He has received financial support in the form of a National Science Foundation graduate fellowship and from both research and teaching assistantships.

The author will join the computer science faculty of the California Institute of Technology as an assistant professor in September, 1981.

This empty page was substituted for a blank page in the original document.

CS-TR Scanning Project
Document Control Form

Date : 8/24/95

Report # LCS-TR-259

Each of the following should be identified by a checkmark:
Originating Department:

☐ Artificial Intellegence Laboratory (AI)
☒ Laboratory for Computer Science (LCS)

Document Type:

☒ Technical Report (TR) ☐ Technical Memo (TM)
☐ Other:_____

Document Information

Number of pages: 220(227-IMAGES)
Not to include DOD forms, printer instructions, etc... original pages only.

Originals are:
☐ Single-sided or
☒ Double-sided

Intended to be printed as :
☐ Single-sided or
☒ Double-sided

Print type:
☐ Typewriter ☐ Offset Press ☐ Laser Print
☐ InkJet Printer ☒ Unknown ☐ Other:_____

Check each if included with document:

☐ DOD Form ☐ Funding Agent Form ☒ Cover Page
☒ Spine ☒ Printers Notes ☐ Photo negatives
☐ Other: _____

Page Data:

Blank Pages (by page number): _____

Photographs/Tonal Material (by page number): _____

Other (note description/page number):

Description :	Page Number:
IMAGE MAP: (1-220) UNTIED TITLE PAGE 2-219, UNTIED BLANK	
(221-224) SCANNING ONLY COVER, SPINE, PRINTER'S NOTES	
(225-227) TARGETS (2)	

Scanning Agent Signoff:
Date Received: 8/24/95 Date Scanned: 8/29/95 Date Returned: 8/31/95

Scanning Agent Signature: Michael W. Cook

Rev 9/94 DSIL/CS Document Control Form cstdform.ved

Scanning Agent Identification Target

Scanning of this document was supported in part by the **Corporation for National Research Initiatives,** using funds from the **Advanced Research Projects Agency** of the **United states Government** under Grant: **MDA972-92-J1029.**

The scanning agent for this project was the **Document Services** department of the **M.I.T Libraries.** Technical support for this project was also provided by the **M.I.T. Laboratory for Computer Sciences.**

Scanned
Date: 8/31/1995
M.I.T. Libraries
Document Services

darptrgt.wpw Rev. 9/94