ADVANCED MIXED-MODE SIMULATION TECHNIQUES

by

J. E. Kleckner

Memorandum No. UCB/ERL M84/48

6 June 1984

Copy

ADVANCED MIXED-MODE SIMULATION TECHNIQUES

by

J. E. Kleckner

Memorandum No. UCB/ERL M84/48

6 June 1984

ELECTRONICS RESEARCH LABORATORY

College of Engineering
University of California, Berkeley
94720

Advanced Mixed-Mode Simulation Techniques

By

James Ellis Kleckner

B.S. (California Institute of Technology) 1975
M.S. (University of California) 1977
M.S. (University of California) 1980

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

Engineering

in the

GRADUATE DIVISION

OF THE

UNIVERSITY OF CALIFORNIA, BERKELEY

Approved: Richard Newton 4/18/84
Chairman Date

4/18/84

4/19/84

Advanced Mixed-Mode Simulation Techniques

by

James Ellis Kleckner

ADVANCED MIXED-MODE SIMULATION TECHNIQUES

James Ellis Kleckner

Ph.D.

Department of Electrical Engineering
and Computer Science

Sponsor: Hewlett-Packard

Signature Richard Newton
A. Richard Newton
Committee Chairman

ABSTRACT

The increasing complexity of integrated circuits has resulted in higher costs for the simulation of those circuits. As a result, designers resort to abstract models to describe the circuit behavior that are less expensive to evaluate than detailed models. This leads to the problem of verifying that the abstract models are consistent with the detailed models they replace. Mixed-mode simulators have been developed which allow the use of abstract and detailed models in the same simulation program, providing faster simulation speeds and the ability to verify the consistency of the abstract and detailed models.

In this dissertation, advanced algorithms and techniques for mixed-mode simulation are presented. Mixed-mode simulation has been extended to allow simultaneous simulation of models at the detailed electrical, logic, and register-transfer levels i.e., those levels that can be represented by schematic diagrams. This investigation shows that mixed-mode simulation of widely differing models can be performed using a program architecture that facilitates the addition of new simulation levels without sacrificing either simulation speed or accuracy.

One result of the investigation is the development of a new simulation program, SPLICE2, which has been used to experiment with program architectures and simulation algorithms. New methods are presented for efficient scheduling of activity in an event-driven selective-trace simulator which are called *cached scheduling*. A new technique for accurate circuit level simulation called iterated timing analysis is also studied. Iterated timing analysis can provide up to two orders of magnitude speed improvement over more conventional forms of circuit simulation.

ACKNOWLEDGEMENTS

The author is deeply grateful to Prof. A. R. Newton, whose foresight and encouragement have made this research both possible and fun. He would also like to express his appreciation to Professors D. O. Pederson and A. L. Sangiovanni-Vincentelli who, along with Prof. Newton, have created a research environment that is very fertile for new ideas and projects.

R. Saleh, K. Keller, T. Quarles, S. Greenberg, J. White, and J. Deutsch, who have contributed substantially to this project with programming, assistance with examples, and numerous discussions, deserve many thanks. The author would also like to thank A. Vladimirescu, E. Cohen, G. Casinovi, E. Lelarasmee, K. Sakallah, and the inhabitants of 321 Cory Hall for many useful discussions. He is further indebted to J. Valdez, S. Greenberg, S. Potter, and B. Murphy for reading the manuscript and improving it with many comments, and to M. Bales for help producing it.

Support received from the Engineering Productivity Division of Hewlett Packard is gratefully acknowledged along with computers and equipment donated by the Digital Equipment Corporation and Tektronix Incorporated.

The author would especially like to thank his parents, Janet and James, for all the guidance, love, and encouragement they have given and his wife, Jennifer, whose love, understanding, and energy has made this experience both more meaningful and enjoyable than had been thought possible.

As usual on reaching the summit, it becomes clear that there are many higher peaks to climb that were not visible from below. So on to new peaks, with new and old friends alike.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

Introduction

Simulation has long been used to verify the correctness of integrated circuit designs. However, increasing size and complexity of circuit design problems has forced designers to accept less and less detailed simulation. Large designs are too expensive to be simulated at the most detailed electrical level such as with SPICE2 [4]. An estimate [5] of the CPU time and memory that would be required by SPICE2 to simulate one 32-bit integer multiply instruction of a recent 450,000 device microprocessor [6] shows that the resources required exceed the capabilities of any computer that exists today. As a result, simulators have been developed to span a wide spectrum of levels of modeling, including behavioral, register-transfer, logic, timing, circuit, device and process simulation. Each simulation level can provide successively more detailed information for more computing cost. It is a desirable extension, therefore, for simulation programs to span more than one level so that models at different levels can be cross-checked. Mixed-mode simulation refers to simulation where more than one model level is used during simulation. The starting point for this work has been the SPLICE1 mixed-mode simulation program [1].

In this dissertation, advanced algorithms and techniques for mixed-mode simulation are presented. For reasons described later, only those types of simulation that can be represented by a schematic diagram i.e., are structural, are examined in detail. The thrust of this work is in two main areas. The first concerns techniques for combining effectively different kinds of simulation into one simulator. This includes issues such as simulator architecture, the representation of signals, and scheduling of activity. The second area is a new technique for circuit level simulation called Iterated Timing Analysis (ITA) [5, 7]. ITA is a relaxation based form of electrical simulation that is faster than standard techniques for simulation of large circuits. These ideas and techniques have been explored using a new simulation program called SPLICE2. To summarize a few key results here, a speedup of one to two orders of magnitude in large electrical simulations is achieved without performance penalty for mixed simulations. All levels of simulation from electrical through register-transfer level are included. Scheduling of activity is done with a floating point time variable using a novel and efficient scheduling technique.

The SPLICE2 program has been developed from a broad base of work [4, 8-14] most notably the SPLICE1 program (for more information on SPLICE1, see [1] and [3]). One goal of this effort has been to create a general-purpose program for the simulation of circuits that can be used to test new algorithms. The algorithms used must work independently of underlying circuit technology and must not be specific to just one technology, such as an n-channel depletion-load MOS process. The specific problem to be solved is that of dc operating point and nonlinear time domain transient simulation of the behavior of circuits at the electrical, logic, and register levels of description.

In the remainder of this chapter some background on simulation methods is presented and the mixed-level simulation problem is specified more fully. In Chapter 2 the issues for the design and architecture of a mixed-mode simulator and the choices made for the SPLICE2 program are presented. In Chapter 3 the electrical simulation problem is introduced and a tutorial example is used to illustrate the fundamental procedures. In Chapter 4 the Iterated Timing Analysis (ITA) method in particular is developed and compared with other techniques. In Chapter 5 discrete simulation at the Logic and Register-Transfer Level (RTL) is covered. In Chapter 6 examples of simulation spanning all the levels are given. Finally, in Chapter 7 conclusions and discussion of directions for further work are presented.

1.1. Simulation

If appropriate electrical circuit models are available for circuit components, the electrical behavior of a circuit can be predicted accurately using computer simulation. Usually, the purpose of this prediction is to verify that a given network of devices will function correctly. The only way to avoid the need to verify circuit function is to use a design method that produces designs that are guaranteed to be correct by the techniques used to construct them (often referred to as *correctness-by-construction*). There is a large amount of active research on this topic but correctness-by-construction is, as yet, only feasible with specialized or constrained design styles. Other methods that are used for solving aspects of the verification problem include timing verification [15, 16] and fault analysis [17]. Timing verification is a technique that is used to verify that clocked logic designs satisfy the constraint that the delays of all paths from clocked elements, or registers, through combinational logic are less than the proper clock phase width. Fault simulation is used to determine how effective a particular input test pattern is at finding faults in the circuit. It is often based on an underlying logic simulator. Testability analysis [18] is a statistical technique used to determine the controllability and observability of the nodes of a network. It gives an estimate of the difficulty of finding a test pattern to detect faults at each node of a network. Simulation is also used as a predictor for optimization-based design of circuits [19].

1.2. Mixed-Mode Simulation

Mixed-mode simulators allow the simulation of devices by combining fundamentally different models and algorithms in the same simulation. The purpose of the following sections is to show where the SPLICE2 program fits in a taxonomy of simulation levels.

1.2.1. Types of Simulation

In the course of design of electronic circuits, many types of simulation are used. Figure 1.1 lists seven kinds of simulation and indicates one typical use for each. Examples of process level simulators are the SUPREM [20] and SAMPLE [21] programs. This type of simulator is used by the process engineer to design or adjust the integrated circuit fabrication process. An example of a device level simulator is the GEMINI program [22]. Device level simulators are used by the process engineer to evaluate a process and develop models that predict the behavior of integrated circuit devices. An example of a circuit level simulator is SPICE2 [4]. Circuit level simulators are used by electrical engineers to check the detailed performance of a prospective circuit or possibly to experiment with new configurations. Timing simulation refers to the approximate electrical analysis such as found in the MOTIS [23] or initial SPLICE1 programs [1] and is used for the performance evaluation of large logic designs. An example of a logic level simulator is the TEGAS program [17]. Logic simulators are used to check the functional correctness of a logic circuit. Examples of RTL simulators are ISPS [24] and ADLIB [25]. Register-transfer level simulation differs from behavioral simulation primarily in that it is more structural with information between blocks communicated on busses. ISPS and ADLIB are RTL simulators as well as behavioral simulators. Examples of behavioral simulators include simulation languages such as GASP [26] and SIMULA [27]. These types of simulators are used by designers to check the functional correctness of algorithms or system components. Sometimes they are used to simulate microcode or perform operating system development.

1.2.2. Schematic Simulation

As stated earlier, the present work deals only with schematic simulation. Schematic simulation is used here to mean simulation of electrical circuits at those model levels for which the circuits can be considered connections of lumped elements, or schematics, where

Level	Typical Use
Behavioral	Algorithm Verification
RTL	Logic Verification
Logic	
Timing	Performance Evaluation
Circuit	
Device	Device Model Development
Process	Fabrication Process Development

Figure 1.1 : Simulation Levels

the communication between devices in the real circuit is implemented using wires. In particular this includes circuit, logic, and RTL model levels. The analysis of devices and processes is not easily represented strictly with lumped elements and wires and therefore is not schematic. The high-level models of behavioral simulation often communicate using abstract data types and via common variables and are also not schematic. The schematic levels of simulation were chosen for the SPLICE2 program since they have a common set of characteristics, as is shown in Chapter 2. Thus, SPLICE2 is intended for use in design from detailed electrical simulation up through simulation of building blocks at the register level.

1.2.3. Previous Work on Mixed-Mode Simulation

A number of mixed-mode simulators have been developed in the past decade [1, 2, 13, 25, 28-33]. The term mixed-mode simulator is used here to mean one which allows the simulation of devices using fundamentally different models and algorithms in the same simulation. Some researchers use the term multi-level simulation to mean the same thing. The term multi-level is not used here to prevent confusion later since logic signal values have multiple levels and multiple strengths. A possibly better term than either of these terms is mixed-level simulation.

Figure 1.2 is a partial list of mixed-mode simulators that shows the choices of simulation levels implemented. The simulation levels from register-transfer through circuit simulation are grouped together as schematic levels of simulation. The last column indicates what method is used in the program to exploit latency (or inactivity) in the circuit being simulated. The bypass method refers to the technique of checking the input variables to a sub-block or element and if they have not changed since the last iteration, the sub-block is not simulated and the previous solution is used. The selective trace method refers to the technique of scheduling a sub-block to be processed only when the input signals change and not even checking it when they do not. Parentheses around the mark indicating that the simulation level is present in the program mean either that the simulation level is not fully implemented

SIMULATOR	SIMULATION LEVEL						LATENCY
	BHV	RTL	LOG	TIM	CIR	DEV	METHOD
MEDUSA (Aachen)[31]					X	X	BY
DIANA (Leuven)[2]*			X	X	X		BY
SAMSON (CMU)[13]			(X)		X		ST/BY
SPLICE1 (UCB)[1]			X	X	X		ST
SPLICE2 (UCB)		X	X		X		ST
M-M SIM. (Bell)[29]		X	X	X			ST
INTSIM (T.I.)[34]	X	X	X				ST
ADLIB (Stanford)[25]	X	X	(X)				ST
MIXS (NEC)[35]	X	X	(X)				ST
Other Func.	X	(X)	(X)				N/A
SHIELD (Hughes)[30]	X	(X)	(X)	(X)	(X)		N/A

BHV	Behavioral Simulation.
RTL	Register-Transfer Simulation.
LOG	Logic Simulation.
TIM	Timing Simulation.
CIR	Circuit Simulation.
DEV	Device Simulation.
BY	Bypass Method.
ST	Selective Trace Method.

* The DIANA program has recently been partitioned into two programs - DIANA-UP for bottom-up design and DIANA-DOWN for top-down design [33].

Figure 1.2 : Mixed-Mode Simulators

or that the method of interfacing is such that the simulation level is not fully integrated into one program.

CHAPTER 2

Architectures for Schematic Simulation

The architectural issues for mixed-mode simulation and the tradeoffs and choices made in the SPLICE2 simulator are presented in this chapter. The topics that follow include such issues as the goals used during the design of the SPLICE2 program, how the target computing environment affects the design choices, which simulation levels should be implemented, schematic simulation and signal representations, how different simulation levels communicate signal values, and finally how to represent time and perform event scheduling for algorithms of widely varying nature.

2.1. Goals

The first step in the design of a mixed-mode simulator is defining the scope of the simulator and its goals. The scope of this work deals with the dc and transient simulation problem (DCTRAN). This type of simulation is applicable to a wide range of simulation levels that are *schematic* in nature. The characteristics that link these levels are explored more fully below in the section on schematic simulation. Briefly, the problem is to find the signal waveforms as a function of time. Other possible types of analysis include small-signal AC, noise, distortion, fault and timing verification. These other techniques are not applicable to as wide a range of simulation levels as the DCTRAN problem is. The goals for the SPLICE2 program are:

(1) Speed of simulation. Some of the major points of this thesis concern speed. It is important that each simulation level of the mixed-mode simulator when used for only that type of simulation not be significantly slower than a simulator of that type that is not mixed-mode. Otherwise, the user will resort to using the individual simulators. The overhead of translation between model levels should only be paid where necessary.

(2) Technology independence. The algorithms chosen should not be limited to one technology. For example, logic simulation of TTL, ECL, NMOS or CMOS should all be possible. At the electrical level, more than just MOS devices should be available. Again, these should be available without speed tradeoff.

(3) Permit varying simulation methods. The simulator should not decide for the user how he or she is to use it. It should provide primitive functions that are powerful enough to be used to build higher level functionality. An example of this is higher level model extraction and verification. A primitive function that takes a computed value from a simulation and stores it in a parameter of the input circuit description is sufficient to allow simulations that automatically extract relevant electrical information for higher level models in a manner that is similar to the actual test jig setup that might be used in the laboratory.

(4) Ease of adding new models. This is an important goal at all levels of simulation but is especially important at the register-transfer level where the blocks are likely to have functionality that is quite different from standard libraries of devices and gates. This is accomplished in SPLICE2 by allowing models to be added at execution time and by using automatically generated tables to allow device models to be read and initialized.

(5) User interface flexibility. The primitives available in the simulator should accommodate powerful and flexible user interfaces, and, as much as possible, the mechanisms by which the user interacts with the simulator should be regular and consistent.

(6) Machine transportability. Within limits, the simulator should be transportable to a variety of computers and computer systems.

These sometimes competing goals are then weighed against one another.

2.2. Computing Environment

In the design of a large program, it is important to take some note of the computing environment in which the program will be used. The stated goal of machine transportability raises the usual questions about generality versus performance. A special purpose simulation algorithm potentially can be made much faster than a general purpose algorithm. A text editor that works with only one type of video terminal can take advantage of more of the features than one that must work with many terminals. The question then arises about how much the computing environment should be allowed to affect the design decisions.

The nature of computing is changing away from the large centralized batch computing environment toward smaller, more distributed interactive computing environments. Computer and disk memory costs are dropping rapidly and good floating point performance is common. Most computing environments are becoming *virtually extended* so that programs have a much larger addressing range than that of physical memory. These trends ease the task of making a program more transportable since detailed knowledge of the underlying machine is not needed. However, the trend toward better human interfaces and work stations with high performance graphics pushes in the opposite direction.

The near-term future will continue to have lower prices for disk and main memory. The use of personal work stations with substantial amounts of main memory will provide more availability of computing but without increasing the peak performance of the computing engine over that of a shared computer. Thus, even more than today, the speed/space performance tradeoff will be biased in favor of speed at the expense of space. Applications will rely more heavily on high performance graphic interfaces for humans. The trend will be toward local-area networks that connect work stations to mainframes for simulations that are large, but the user will still want to interact with the mainframe via the work station. Thus, it is important for time consuming applications, such as simulation, to be kept portable to the large computer environment.

2.3. User Interface

To take advantage of advanced graphic interface packages, while still maintaining large batch capability, the front and back ends of the simulator can be made separable so that they communicate via a set of functions. In the SPLICE2 program, the simulator is viewed as a net list processor (NLP) by the program that is the user front end interface. The set of functions are defined in the NLP [37] specification and are used to transfer the information about models, elements and their connectivity from the front end program to the simulator. Currently, there are two implementations of NLP driver programs for SPLICE2, a textual driver (the BLT program [38]) for compatibility with SPLICE1 and SPICE2, and a graphic editor program used at UC Berkeley for integrated circuit design and schematic diagram generation (the HAWK program [39]) Figure 2.1 shows how SPLICE2 communicates with the front end via NLP. The simulation results are written using a waveform storage package (WAP) [40] that can be monitored interactively.

Control and input for the SPLICE2 simulator are merged by using special simulation elements called control elements that cause specific control actions to occur when the element is processed by the scheduler. This is similar to using object-oriented or data-flow programming

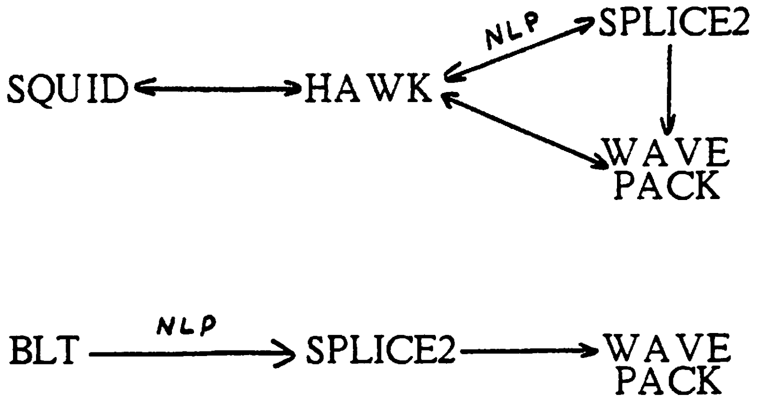


Figure 2.1 : User Interface

methods where the functional or data-flow program is the circuit that is made of control elements. Control is accomplished by creating special elements and setting their parameters in exactly the same way that ordinary elements are created. This results in an extremely uniform interface between the front end program and the simulation engine. Thus, execution time options are passed as parameters to an option control element and the end of simulation occurs when the *end* control element is processed. In a similar fashion, interactive breakpoint elements are available to suspend the simulation at specified times or on simulation generated events. In this way, greater functionality can be added for the interactive environment without affecting.the ability to run in a large batch environment.

2.4. Adding New Simulation Models

New simulation models are added to the SPLICE2 program by allowing the user to program the new element behavior in the C programming language, compile the new model using the C compiler into machine executable object code, and then load (sometimes called link) the new machine code with the SPLICE2 program. This technique of extending and using an existing programming language for writing user generated model code is sometimes called an *embedded* approach. The details of creating a model for the SPLICE2 program are given in Appendix D. Briefly, the user must provide three types of information to the simulator for a new model: the topology (or net list information), the parameters, and the executable code. The topology and parameters are specified in a table that is built by a preprocessor which reads the C language data structure declaration for the element. The declaration has some added keywords for indicating the name, simulation level, and direction for each terminal for the element. The keyword PARAMETERS is used to indicate which members of the data structure are to be filled in as parameters by the user at run time. Thus, the member name and type given in the declaration of the data structure are automatically used as the name and type of the parameter for reading and initializing. In this way, there is very little chance of error in naming and using the parameters. Thus, adding a model requires only the declaration of its C programming language data structures and the function which implements the model. An optional initialization function can be provided if any pre-calculation of values is needed for speed (such as the critical voltage for diodes [4]).

Dynamic loading is a technique for adding executable code to a running program and is found commonly in LISP programming environments [41]. SPLICE2 has a dynamic loading capability which, in combination with the table mechanism for model declaration presented above, allows new models to be added easily to a simulation even after execution of the simulation program has begun. Dynamic loading is not machine transportable and usually needs to be re-programmed for new computing environments and thus is at odds with the goal of machine transportability. However, conditional compilation makes it possible to load models statically and simulate when dynamic loading is not available. Libraries of models compiled into the simulator are adequate for electrical and low-level logic simulation, but at the register-transfer level (RTL), the user must be able to add new functions easily. RTL simulation is described in more detail in Chapter 5.

2.5. Inter-Simulation Communication

One basic decision about the simulation architecture that must be made at the outset is how the different levels of simulation communicate with each other and how much each level of simulation must know about the other levels of simulation. Various approaches exist, such as the SHIELD program [30] which runs each level of simulation separately with each simulation algorithm advancing in time only so far as its inputs allow. This allows each simulation program to be optimized for its type of simulation and keeps all the knowledge about how to translate from one level to another isolated in the control program. The primary difficulty with this approach is that the overhead is high when several levels are used and components of differing type are intermingled arbitrarily. A copy of every interface signal must be kept and transmitted through the supervisor program. Also the overhead of the scheduler is incurred more than once for each simulation event since each simulator performs time control locally as well as in the supervisor and must check to see when to return control to the supervisor.

The design choice for the SPLICE2 program has been to use one integrated scheduler mechanism for all simulation levels. Algorithms have been chosen and developed that use signals in a consistent way so that translation of a signal from a more detailed level to a less detailed level results in a reduction of detail that preserves the maximum amount of information. The design of the scheduler and the signal representations are described more fully later in this chapter.

2.6. Choice of Simulation Levels

Figure 2.2 gives an estimate of the CPU time and memory that would be required to simulate one 32-bit integer multiply instruction of a recent 450,000 device microprocessor [6] using four different levels of device models [5]. The multiply requires 1.8μs in 33 clock cycles of 55ns each. These estimates are based on experience with the SPICE2 circuit analysis program [4, 42] and the SPLICE1 mixed-mode simulator [1, 3]. The CPU times shown are normalized to the performance of an IBM 370/168 and ignore virtual machine overhead i.e., assumes uniform addressing and memory access times. The term circuit means accurate analysis such as found in the SPICE2 program. Timing refers to the approximate electrical analysis such as found in the MOTIS [23] or SPLICE1 version 1.3 programs [1]. For the purpose of this work, *logic* means discrete valued simulation of unidirectional elements and *RTL* means *register-transfer level* where the elements may use complex internal models, such as an arithmetic logic unit. These model levels differ mainly in the way that signals are represented and combined, as is described later.

Figure 2.2 shows how reducing model complexity reduces the simulation cost. Two effects are responsible for this reduction. The first is that simpler models require less CPU time to evaluate. The second is that higher-level models represent the functionality of a number of lower level models and therefore require less data storage and fewer evaluations.

LEVEL	CPU TIME	$\log_{10}$ (CPU)	MEMORY	$\log_{10}$ (MEM)
CIRCUIT	6 Months	7.2	250 MB	8.4
TIMING				
ITA	12 Hours	4.6	60 MB	7.8
NTA	8 Hours	4.5	30 MB	7.5
LOGIC	10 Min.	2.8	4 MB	6.6
RTL	<<1 Min.	1.8	<<.5 MB	5.7

Figure 2.2 : Simulator Performance

The logarithmic values for time and memory indicate about one order of magnitude reduction in storage per level and about two orders of magnitude reduction in CPU time. The CPU time reduction reflects the effect of both reduced device count and simplified models.

The levels of simulation chosen for implementation in the SPLICE2 program are circuit, logic and RTL. The circuit simulation level uses a new method called iterated timing analysis (ITA) [3, 5] that is based on the non-iterated timing analysis (NTA) [1] from the SPLICE1.3 program with the relaxation iteration carried to convergence. ITA requires approximately 50% more CPU time than plain timing analysis but the actual requirements depend on the characteristics of the circuit being analyzed. It provides dc operating point and voltage waveforms with accuracy comparable to that of SPICE2. ITA is presented more fully in Chapter 4.

The choice of circuit, logic and RTL is only one choice for a set of simulation model levels. More detailed simulation such as device modeling and more abstract simulation such as behavioral simulation are not included in SPLICE2 because the algorithms are not similar enough to be compatible with the other three. Circuit, logic and RTL are similar in that circuits of those levels are represented by schematic diagrams. Figure 2.2 suggests that the choice of ITA, logic and RTL spans the range of schematic simulation fairly evenly. The difference in run time between ITA and logic analysis suggests that another form of approximate electrical simulation between ITA and logic might be appropriate. The inclusion of each additional model level adds more overhead in translating to other levels and makes the program bigger and more complicated. This overhead and complexity must be offset either with improved accuracy or faster overall simulation performance due to the hierarchy of model levels (higher level models require less time to evaluate than lower level models). The large fixed cost of adding a model level suggests that ITA, logic, and RTL are sufficient. In Chapter 5, it will be seen that a more general logic model can be used in the same way as a simple logic model and run with speed comparable to the simpler model. Some simulators allow the simultaneous simulation of logic elements that have models that use different numbers of logic states by allowing arbitrary numbers of translation routines. This approach can be expensive both in CPU time and complexity of the program.

2.7. Schematic Simulation

Schematic blocks are modules that communicate along wires. Figure 2.3 illustrates the structural relationship between blocks and wires. This partitioning suggests an architecture where blocks are represented by routines that model their behavior and wires (nets or nodes) are elements that understand how to transmit and translate the signals. This results in a simple architecture of one data structure for topology and state and one model routine for each element. The wires have model routines that implement each level of simulation. Thus there are electrical, logic, and RTL net models and their corresponding signal types. Adding a new level of simulation amounts to defining the new signal type, adding the new net model routine, and updating the other net models to coerce to and from the new signal type. It is this coercion overhead that limits the number of different simulation levels.

Figure 2.4 shows a situation where the net model translates from an electrical signal to logic and register levels. As integrated circuit switching devices become faster and faster, more of the total delay is due to the wires. Having an explicit net model allows these effects

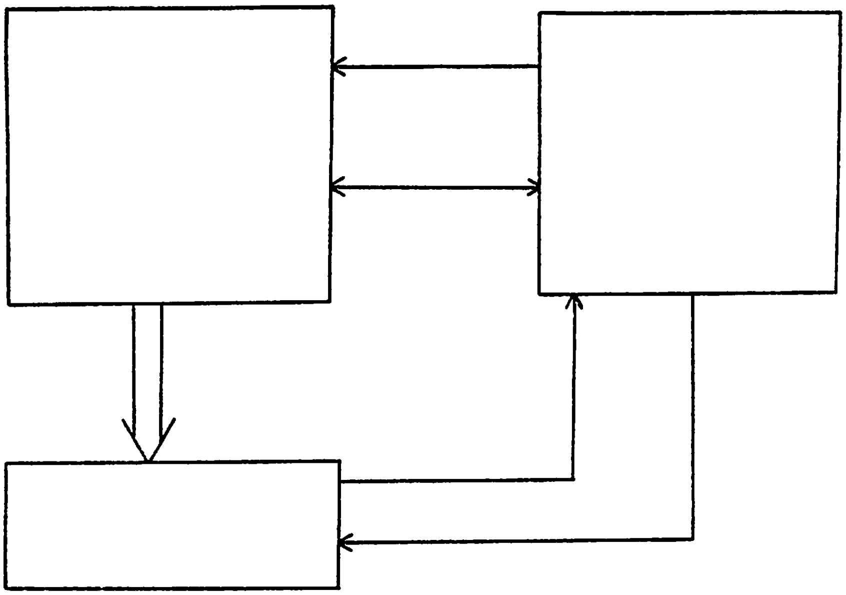


Figure 2.3 : Schematic View of Circuits

to be treated accurately. The representations of signals and the method of combining them is critical for accurate simulation. The signal types in the SPLICE2 program are presented in the next section and the mapping from one to another is shown.

2.8. Signal Representations

A consistent representation for signals over all model levels is critical for accurate mixed simulation. The main difference between model levels is the level of detail used in the models and signals. A high level model uses fewer bits of precision than a more detailed model. This is not to say that the higher level model is less accurate. The fact that the voltage on a wire is 4.285 Volts adds no useful information if the question being asked is simply whether the signal is true or false. Excess precision can easily get in the way when it is not appropriate and *data hiding* is important in hierarchical abstraction.

Since the function of nets is to combine signals, signals must have not only a *level*, but also a *strength*. For steady-state analysis, a Norton or Thevenin equivalent is needed for the

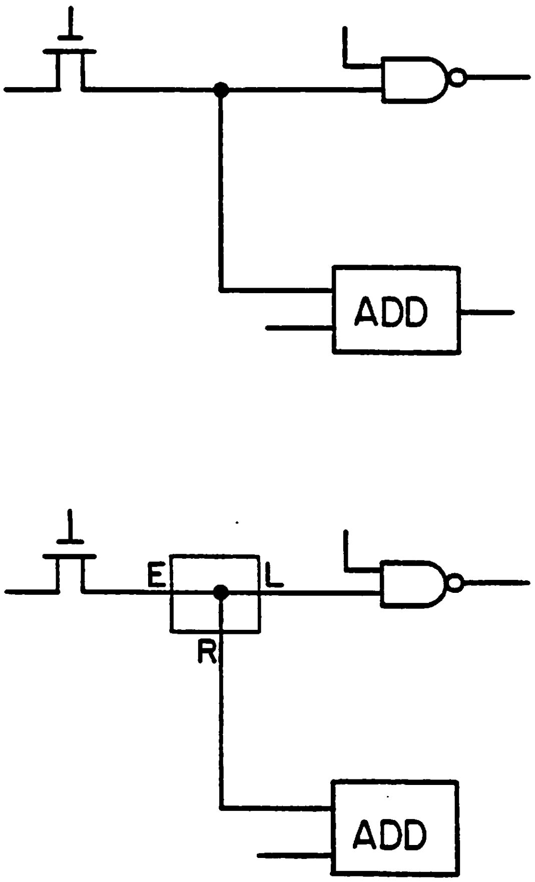
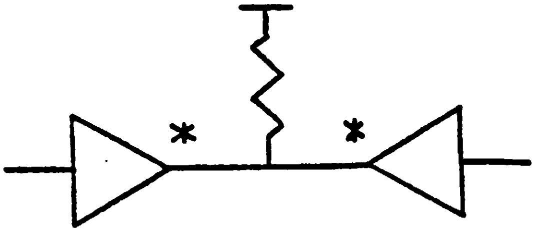


Figure 2.4 : Net as an Element

contribution of each element [43]. Figure 2.5 shows a pair of open-collector drivers that are connected to a bus. This wired-and function is a common example of how signals are combined. Since elements of all types can be connected together, information on how to combine them all must be available through the strength. The difference is that the number of bits used to represent the signal goes down as the detail decreases. The signal level represents the current state of the wire. At the electrical level it is a floating point number for the voltage while at the logic level the number becomes a small integer. Register signals are reduced to binary values that may be collected into arrays in order to represent busses as a single number. A strength bit-array is still necessary to indicate which bits are being driven.



Signal =	Value + Strength
Circuit:	Floating Point
Timing:	Integer
Logic:	Integer
Register:	Bit Array

Figure 2.5 : Signal Representations

2.9. Signal Mapping

Communication between models in a mixed simulation requires translation of signals. At least one function to map signal types in each direction is needed with the addition of every model level. Figure 2.6 shows an example of possible mappings to and from the circuit level, or continuous domain, into the logic level, or discrete domain, and to and from logic into register. The choice of thresholds need not be uniform, but must cover all possible values. The same considerations apply to the discrete mappings. The number of states and the assignment of thresholds clearly depends on how much accuracy is desired and the technology being simulated. More detail on signal mapping is presented in Chapter 5 where discrete simulation is described.

2.10. Time Representation and Scheduling

One of the most challenging tasks in designing a mixed-mode simulator is how to reconcile the divergent requirements of different algorithms for the representation and

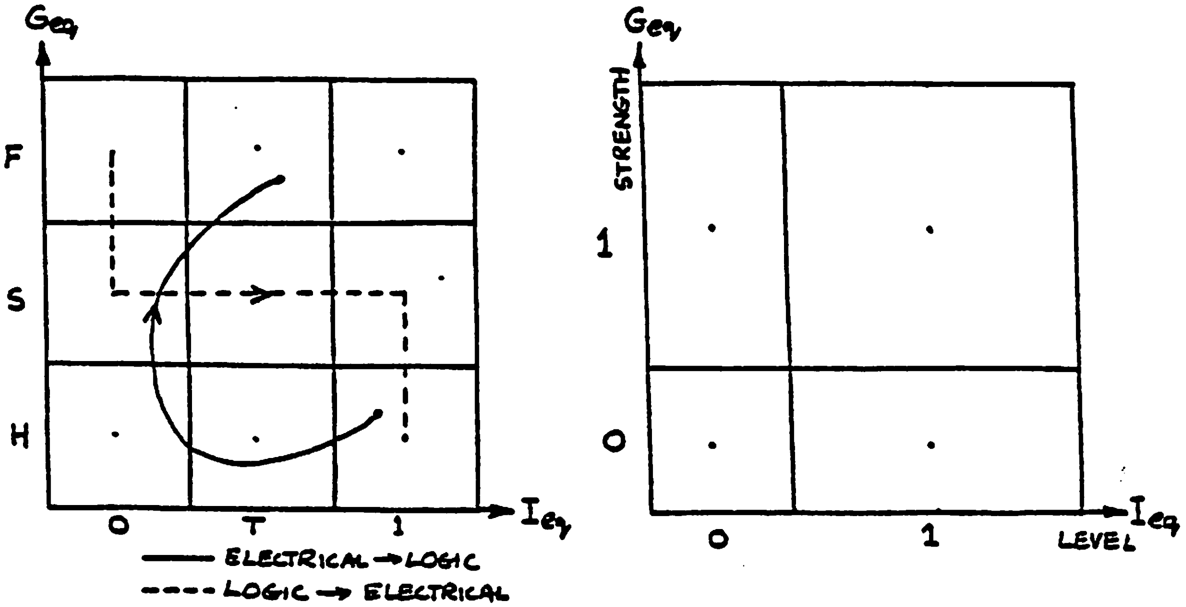


Figure 2.6 : Signal Mapping

advancement of time. The purpose of this section is to define event scheduling, outline some existing event scheduling techniques, resolve the issue of how to represent time, and present some results on a novel approach implemented in the SPLICE2 program called *cached* scheduling.

2.10.1. Event Scheduling

Discrete event simulation refers to computer simulations in which events occur at discrete, arbitrary points in time. The management of time-flow consists of sequencing the events and executing (or processing) the events and propagating their effects. Figure 2.7 illustrates the principal activities of event scheduling. The *event processor* carries out whatever action the event is intended to cause and the *scheduler* sequences the events and controls the evolution of time. The two main functions are *schedule* an event at time t in the future, sometimes called the *hold* function, and *next event* for finding the next action to perform.

Depending on the simulator, there may also be an *unschedule* function to remove a pending event from the queue.

In circuit level simulators such as the SPICE2 program [4], the current value of time is represented by one floating point number that is the same for all of the equations. In logic simulation, such as in the VOTE program [44, 45], time is represented by an integer value and all delays must be specified as a multiple of the unit of time. Scheduling is done via the *time mapping* technique. A good description of the time wheel and the time mapping methods for time control is given in [46]. A technique similar to time mapping is used in the SPLICE1 program [1]. General purpose simulation languages such as SIMSCRIPT [47], GPSS [48], GASP [26], SIMULA [27], and others use a list of events ordered by a floating point time variable. A method for speeding the insertion of events onto the list called the indexed list method is presented in [49] which also gives a good comparison of scheduling techniques. Another variant of the indexed list method is described in [50]. The indexed list method with floating point time is used in the SAMSON program [13]. Floating point time is also used in the

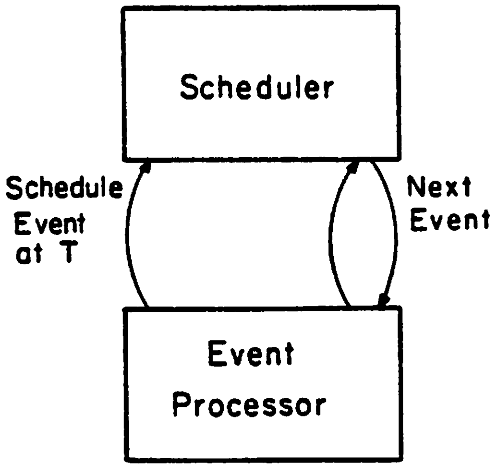


Figure 2.7 : Event Scheduling

ADLIB/SABLE program [25].

In an electrical level algorithm, there is a relatively large amount of computation done when a solution is obtained at a time-point and so the scheduling overhead is not very important. However in logic analysis, relatively little computation is done for each solution. Thus, the scheduling method must be fast and robust in the presence of large numbers of events if it is not going to limit the performance of the simulator.

2.10.2. Representation of Time

A mixed-mode simulation environment requires a large amount of range and precision from the time representation. The time-steps required during the solution of electrical equations may be considerably smaller than the smallest time-step of interest to the user. Therefore, the unit of time may be as small as $10^{-12} - 10^{-15}$. The total simulation time can be orders of magnitude larger than the rise and fall time of any particular logic gate. To be able to represent total simulation times on the order of 1 second yet still resolve electrical events requires a range of $\approx 10^{15}$ or $\approx 2^{50}$. Thus the usual 32 bit integer with a range of $\approx 10^{9}$ is not sufficient. A 64 bit integer has a range of $\approx 10^{19}$ and would allow simulations of over one hour of simulated time with a resolution of 10^{-15} seconds. Unfortunately, common programming languages do not have intrinsic 64 bit integer data types (even though some computers such as the VAX support it). Functions could be added to perform the extended arithmetic, but the overhead of calling those functions would be quite large. A single precision floating point number usually has a mantissa with a size less than 25 bits. Thus it has inadequate precision to resolve small time-steps when the time becomes large. A double precision floating point number usually has a mantissa of size greater than 50 bits and thus has the required precision. On computers with floating point units, the overhead of using a double precision number for time is likely to be less than the function call overhead of using an extended integer. An argument can be made that extended integers have a place as a scalar type in computer languages.

One disadvantage of the floating point representation for time is that the finite mantissa size causes arithmetic operations not to be commutative and associative. Thus, the exact result of a calculation depends on the order in which the operations are carried out. An advantage of the extended integer is that it can be used directly or bits extracted for use with the scheduler index. Tom Quarles has pointed out [51] that if care is taken to prevent overflow of the mantissa, then the floating point number can be used for exact calculations. This is similar to using the floating point number as an integer of size equal to that of the mantissa and is what is done in the SPLICE2 program. A double precision variable is used to represent time and two C language macros are defined: TO_NORM_TIME() to convert simulation time to normalized form and TO_WALL_TIME() to convert normalized time to simulation time. Addition and subtraction of normalized times can be performed safely but computations involving multiplications and divisions require re-normalization of the time variable. Because of the use of a C language type definition for time, if a 64 bit integer were to become available in the C language, only the type definition and the macros would need to be changed.

2.10.3. Indexed List Method

A simple organization for ordering all events by time is a linear linked list. If events are likely to be canceled, a back link is useful for quick removal from the list. For sequences of events with uniform distributions, the average number of scans for insertion on a linear list is $\frac{N}{2}$ where N is the number of events already on the list. Thus, the total number of scans to insert N events is proportional to N^2. The indexed list method mentioned above reduces the number of scans by dividing the list into smaller sub-lists each of which spans a sub-interval of time. Figure 2.8 shows the data structure for the indexed list method. Dummy events are created as time markers to head each sub-list. The sub-intervals usually span equal sized time bins, although nonuniform spacing can be used. With uniform intervals, insertion of a new event is done by computing the index into an array of pointers to the marker events and then linear insertion of the event on that sub-list. The index interval size must be adjusted periodically so that the number of events in each sub-list is neither too few, which means that excessive marker events are allocated and processed, nor too many, which means that insertion will degenerate to the linear case. For large N and uniform event distributions, the speedup of indexed list insertion over linear list insertion will be equal to the number of index bins.

The indexed list method has been implemented in the SPLICE2 program and measurements made to determine the overhead of the marker events and the insertion time for linear lists. These timings and all of the other performance figures quoted in this dissertation were made on a Digital Equipment Corporation VAX 11/780 with floating point accelerator. Figure 2.9 is a plot of execution time versus the number of dummy events in the list. The intercept

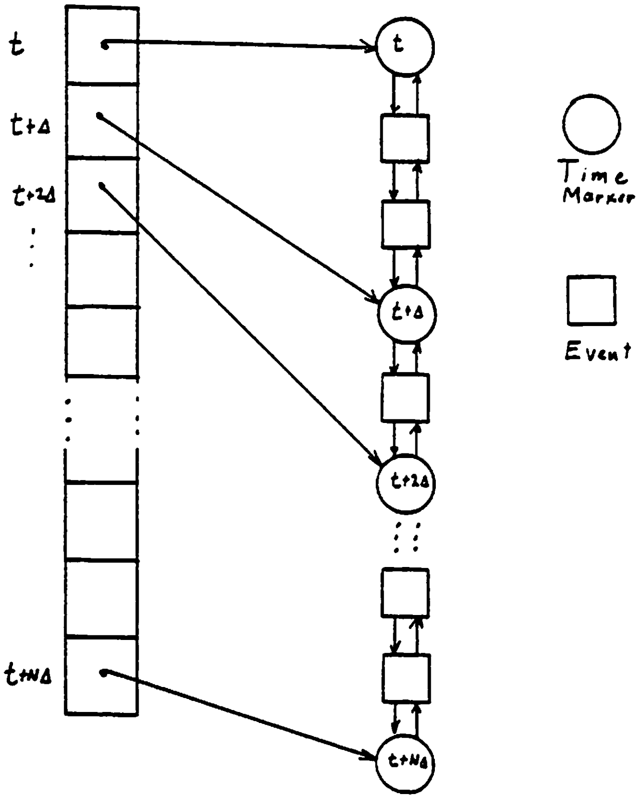


Figure 2.8 : Indexed List Method

at zero is labeled "b" on the graph and indicates an overhead of 39 μS per marker event. This number is a lower bound and is made up mostly of the function call and return time plus the time to increment the current index of the array. If a normal event execution time were m times longer than the marker event execution time and the fraction of total events which are marker events were f , then the execution overhead events would be $\frac{f}{f + (1 - f)m}$. For $f = \frac{1}{2}$ and $m = 4$ the overhead would be 20%.

Figure 2.10 shows the time required to perform linear list insertion as a function of the number of events already on the list with the slopes and intercepts of the curves labeled "b"

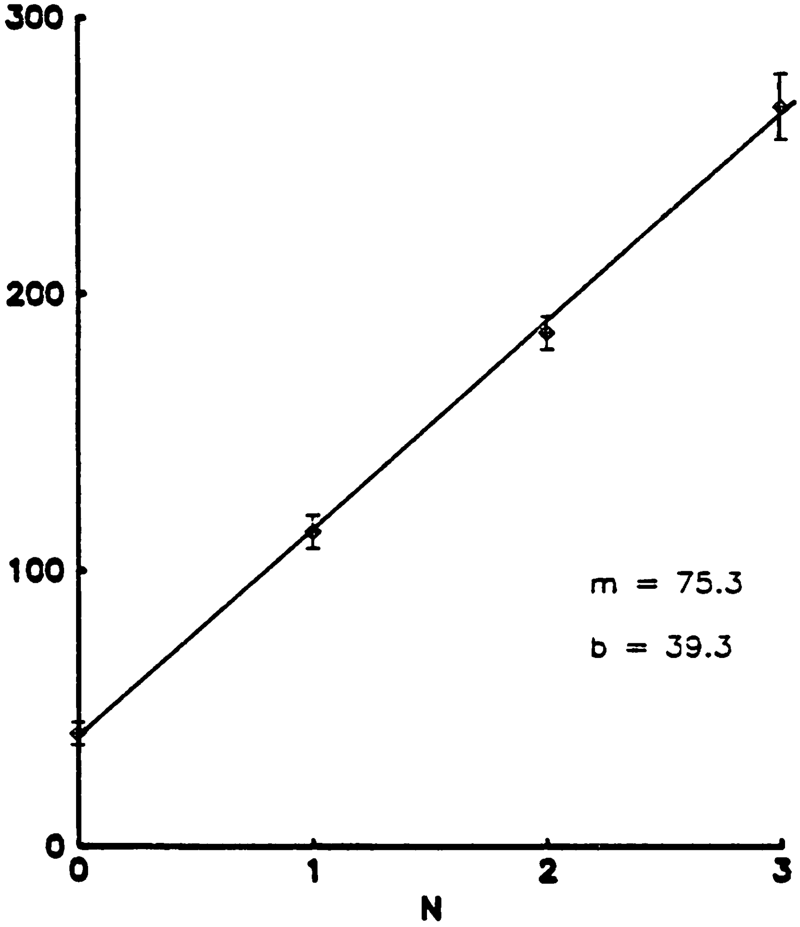


Figure 2.9 : Marker Event Overhead

and "m", respectively. The upper curve is for the indexed list method using a single precision floating point number and the lower curve is using a 32 bit integer number. The intercepts at zero show that floating point scheduling requires about 30% more time than integer scheduling however the slopes are the same to within the accuracy of the measurements. Thus the increase in CPU time for using floating point numbers corresponds to slightly more than the amount of CPU time to scan one extra event. As the number of events increases, this difference becomes less important.

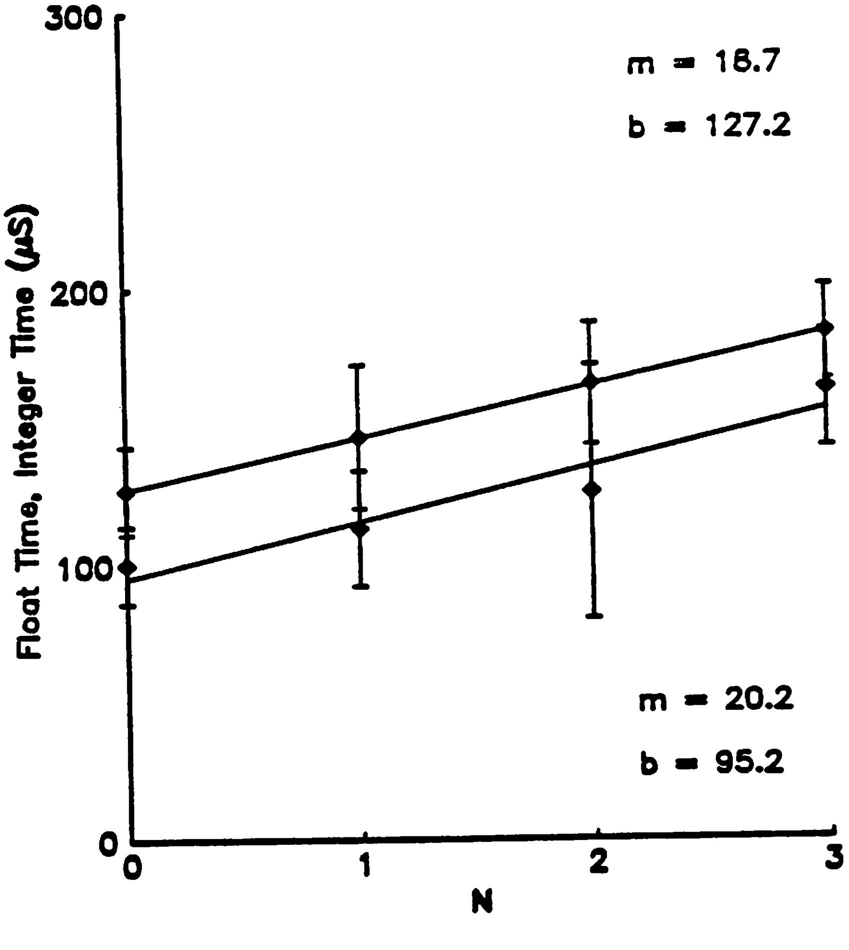


Figure 2.10 : List Insertion Time

The indexed list approach has several disadvantages. First, the dummy events require storage and, depending on details of the implementation, must be allocated and freed. Second, the dummy events lengthen the linear event list by their presence and require CPU time to be processed, thereby limiting the peak speed of the simulator. Third, schemes to re-compute the index interval for the array are required which can lag behind changes in the distributions of events and be fairly involved to carry out. Fourth, backing up simulation time is difficult. This situation occurs in electrical analysis when a time-step is rejected and the interval must be re-simulated with a smaller step size.

2.10.4. Cached Scheduling

Cached scheduling is a new technique for indexing events which makes use of the statistical properties of the time distribution of event scheduling operations. Both the indexed list and the cached list methods use indices to find a *good* starting point in the linear list of events from which to insert the event to be scheduled. The cached list method makes use of the fact that the most recently scheduled event often is likely to be a good guess for the point at which to insert the next event. This is similar to the least recently used (LRU) policy for determining which memory buffer to invalidate in hardware cache memory management.

The method requires keeping one or more *cache pointers* to events. A cache pointer is *valid* if it points to a scheduled event whose scheduled time is less than or equal to the time at which the new event is to be scheduled. Before beginning insertion at the head of the list, the cache pointer is checked for validity and if it is valid, it is used as the starting point for the linear insertion. If the cache pointer is valid, then the event pointed to is at least as good a starting point as the head of the list. After inserting the new event on the list, the cache pointer is set to point to the new event.

Measurements have been made on a two element cache which has been implemented in the SPLICE2 program. One of the cache pointers is reserved for events which are scheduled at the current value of simulated time. When the current time cache pointer misses, then scheduling is performed using the other cache. The measurements are summarized in Figure 2.11. The peak performance is seen to be better than the peak performance of the indexed list technique as compared with Figure 2.10. Note that time is represented as a double precision floating point number for these measurements. The most important number for estimating cache effectiveness is the hit ratio which is the ratio of the number of times that the cache is found to be valid to the total number of schedules using that cache. This simple scheduling method is adequate for electrical analysis and some logic analysis. The overhead becomes fairly high in the Logic3 example indicating that the two element cache is overloaded.

A generalization of the cached list method is the *cached indexed list* method. The cached indexed list method uses an array of cache pointers very similar to the array of pointers to dummy events that is used in the indexed list method. The intention is to add more cache pointers and use the indexing mechanism to make the loading more uniform in order to increase the hit ratio. The algorithm proceeds as follows. When a new event is to be scheduled, the time is used to index into the array of cache pointers. If the index is larger than the array, then the *remote event* action, which is described later, is performed. If the cache pointer is valid, then it is used as the starting point for linear insertion and the cache pointer is updated to point to the new event just inserted. Otherwise, the cache pointer array is searched backwards to find the first valid pointer and that is used as the starting point for linear insertion. If none of the cache pointers are valid, the head of the list is used. The original cache pointer indicated by the indexing operation is updated to point to the new event just inserted.

One observation is that the correct operation of the scheduling procedure does not depend on the validity or efficiency of the cache. In fact, all of the entries could be set to a null value (flushed) every scheduling operation and it would still work. Also, the efficiency of the cache is increased as the loading is made uniform. A cache pointer becomes invalid by one of two mechanisms: first, the event becomes processed and passes into the past, or second, the event is

Example	NEN	NLN	NRN	%CPU	CHR	CAIT (μs)
Elect1	169	0	0	5.0	0.931	157
Elect2	223	0	0	2.7	0.932	118
Elect3	169	0	0	8.0	0.919	271
Logic1	1	153	0	10.5	0.702	319
Logic2	1	61	0	5.2	0.596	205
Logic3	1	235	0	39.5	0.762	423
Mixed1	289	61	0	7.1	0.923	227
Mixed2	1	118	4	28.3	0.691	292

Example	First Cache			Current Time Cache		
	TNS	HR	AIT (μs)	TNS	HR	AIT (μs)
Elect1	107816	0.645	416	463524	0.998	96
Elect2	58033	0.804	162	127533	0.99	96
Elect3	801182	0.799	535	1253854	0.996	102
Logic1	6209	0.603	411	2869	0.918	94
Logic2	3801	0.487	234	1611	0.852	106
Logic3	270396	0.668	582	135211	0.95	113
Mixed1	123892	0.602	812	534643	0.997	90
Mixed2	87077	0.585	363	41220	0.914	115

NEN	= Number of Electrical Nets
NLN	= Number of Logic Nets
NRN	= Number of Register Nets
%CPU	= Percent of total time spent scheduling
CHR	= Composite Hit Ratio
TNS	= Total Number of Schedules
HR	= Hit Ratio
AIT	= Average Insertion Time
CAIT	= Composite Average Insertion Time

Figure 2.11 : Cache Performance Measurements

un-scheduled. The first mechanism is not important since new events are not scheduled in the past. Either there are other events at the current time and the first entry of the cache points to the last one of them, or the head of the list is the best starting point for insertion.

In steady-state operation with all of the cache pointers valid and a uniform distribution of events in time, the number of linear scans for insertion is comparable to that for the indexed list method. A uniform random distribution has no locality of reference and therefore is a difficult test example. With a uniform index interval and a uniform distribution, the average number of events per sub-list is equal, say N . The indexed list method will always start at the beginning of the interval and scan to the position of the new event which will be $\frac{N}{2}$ positions down the list on the average. With the cached indexed list method each cache pointer will point to the middle of its interval on the average. The probability of a cache hit will be $\frac{1}{2}$ and if it occurs the number of scans will be $\frac{N}{4}$ on the average. If the cache hit does not occur, then the number scans from the preceding cache pointer will be $\frac{N}{2}$ to get to the beginning of the current interval and $\frac{N}{4}$ to get to the location in the list in the current interval. Thus the average number of scans will be $\frac{N}{2}$ which is the same as the indexed list method for the uniform distribution.

The main advantage of the cached indexed list method is that there are no dummy events to maintain and re-index. This makes it very simple to back up time which is occasionally necessary for electrical algorithms as has been pointed out. To change the index interval, all that is necessary is to change the index interval variable to its new value. The array of cache pointers need not be altered although it might be advantageous to null them or copy some. The index will automatically achieve equilibrium as events are scheduled and the cache pointers are *faulted in.*

The remote list is managed using a cache with an index interval ten times larger than the main cache. Overflow from the remote index is handled with a final remote list pointer similar to that used in the indexed list approach. The index interval is computed by choosing a value that maintains the number of events in the near and remote sub-lists at some ratio. Other schemes for computing the index interval could be used.

Both the indexed list method and cached indexed list method can have long insertion times when the index is overloaded with events or for pathological event distributions. The technique described by Wyman [50] can be used to improve the worst case performance of the scheduler for both methods at the cost of adding more dummy events. Also, if the list is doubly linked, as is the case in the SPLICE2 program, then insertion scans can also be done in the reverse direction. This would avoid the need to scan events in the previous interval in order to arrive at a position in the beginning of the current interval.

Another way to distribute the loading on the cache is to use auxiliary information about the events such as whether they are electrical or discrete. A separate set of cache pointers could be used for the different types of events. A cache of size equal to the number of buffers used in the windowed iterated timing analysis algorithm works perfectly since electrical events are not scheduled at other times.

CHAPTER 3

Electrical Simulation

Electrical analysis of circuits is used to predict their accurate waveform behavior and is the most detailed level of simulation in a schematic simulator. The present chapter provides the background with which to compare the Iterated Timing Analysis (ITA) algorithm, described in detail in Chapter 4, with other existing techniques. In Section 3.1, the nonlinear dc and time-domain transient (DCTRAN) problem to be solved is defined. A specific circuit example is used in Section 3.2 to show how the nonlinear time-varying circuit equations are solved. In Section 3.3, a circuit interpretation for iterative processes is presented and some of the stability properties are examined. In Section 3.4, the differences between existing techniques for detailed electrical simulation are outlined. Finally, error measures and potential speedup factors are presented in Sections 3.5 and 3.6.

3.1. The DCTRAN Problem

The DCTRAN problem is defined as finding the voltages as a function of time on all of the wires of a given network. Figure 3.1 illustrates this for an example of a chain of inverters. The problem then is to compute an approximate voltage waveform for all of the wires that is *close enough* to the exact solution. The meaning of *close enough* is examined more fully in section 3.5.

The elements of the network are modeled by mathematical equations called the Branch Constitutive Equations (BCE). Together with Kirchhoff's current and voltage laws, these models result in a system of first order nonlinear, time-varying, ordinary differential equations (ODE) which must be solved to predict the electrical behavior of the network. Most circuit simulation programs solve this system of equations by slicing the waveforms into discrete time intervals, or *time-steps*. Circuit elements, such as capacitors and inductors, whose branch

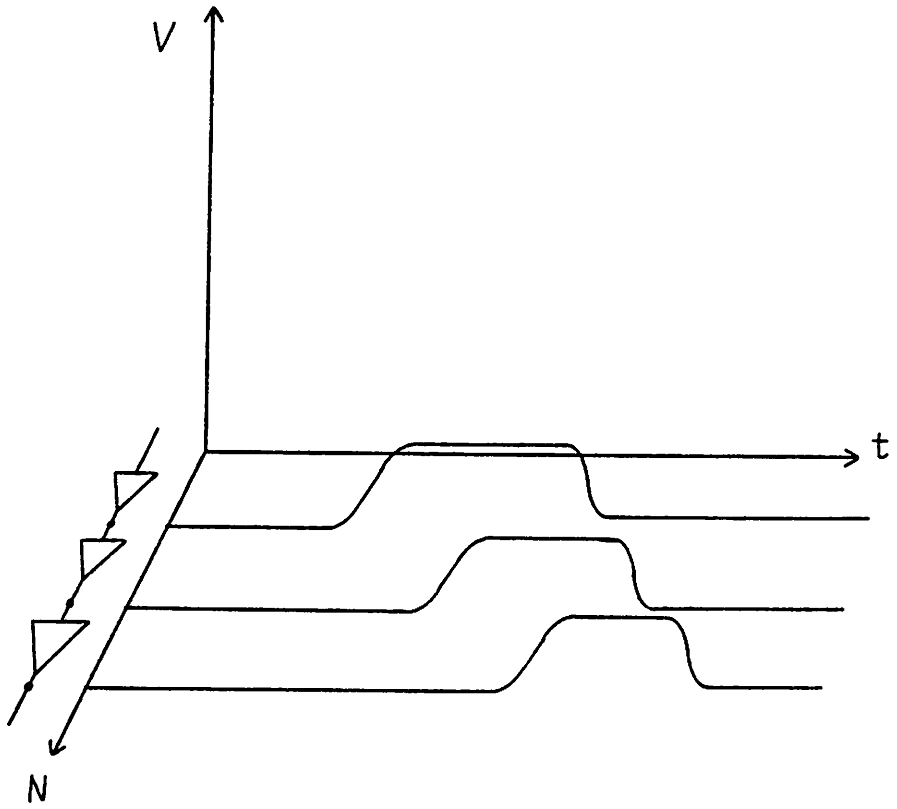


Figure 3.1 : DCTRAN waveforms

relations contain the time variable are said to have *memory*. Using a numerical integration scheme, these elements are then modeled with equivalent *companion* models which, at any given instant of time, are characterized by equations similar in form to the memoryless elements [4, 52]. This leads to a sequence of equivalent problems which, when solved, result in the desired waveforms. For practical numerical integration methods, these waveforms converge to the exact solutions of the ODE's as the time-steps are taken sufficiently small. Another class of methods, known as *waveform relaxation,* [53] has recently been studied which does not decompose the time behavior of the network into a sequence of dc-equivalent problems. Instead, the solution to each equation is approximated for all time and the resulting waveforms are then iterated. In contrast to the waveform approach, the other way of iterating is sometimes referred to as *point at a time*.

In the point at a time approach, each system of nonlinear equations is solved using an iterative technique. In the next section these ideas are demonstrated using the specific example of a resistor, a capacitor and a diode.

3.2. Resistor-Capacitor-Diode Example

Figure 3.2 shows a nonlinear circuit which will serve as a simple example to illustrate the solution process. It consists of a capacitor, a resistor, and a diode with the initial condition on the capacitor that it is positively charged to a voltage of 2 Volts. The procedure here will be to simplify the example problem to the point where it can be solved and then show how the more general solution is built up out of the lower level techniques.

The solution of the DCTRAN problem for this network results in the voltage waveforms for nodes 1 and 2 which are shown in Figure 3.3. The waveforms have a fast and a slow component. Figure 3.3a shows the fast initial transient which occurs when the diode

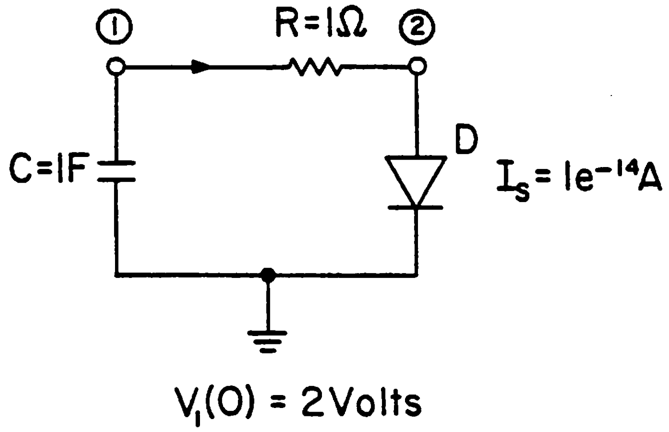


Figure 3.2 : Simple Example

is fully on and has a very high conductance. As the voltage on the diode reaches its critical voltage, the conductance begins to drop quite rapidly, leading to the very slow tail shown in Figure 3.3b.

This example contains most of the characteristics of more complex examples. It has a strong nonlinear element (the diode), it has two nodes requiring a system of equations to be solved, and the capacitor introduces an element with memory. The following subsections will develop general DCTRAN analysis methods by first using this example to demonstrate nodal analysis of nonlinear dc circuits and then showing how integration techniques are used to obtain the time evolution.

3.2.1. DC Solution

The problem as defined here is to find the voltages on all of the nodes of the circuit for the simulated time interval. As mentioned above, this is usually accomplished by breaking time into intervals, or time-steps, and solving a series of dc-equivalent problems which result in the desired solution. In turn, the nonlinear dc problem is solved using an iterative technique (usually Newton Raphson) in which the circuit is approximated by a linear equivalent circuit. In this section, the problem is reduced to one equation in one unknown and techniques for solving it are explored. The generalization to systems of equations can be carried out in different ways and will be examined more fully in Section 3.4 when existing techniques are compared.

Nodal analysis is commonly used for the solution and makes use of the Branch Constitutive Equations (BCE or sometimes just branch equations), Kirchoff's Voltage Law (KVL) and Kirchoff's Current Law (KCL). This is often written in matrix form as

$$\mathbf{Y}\,\mathbf{v} = \mathbf{i}$$

where $\mathbf{Y}$ is the nodal admittance matrix, $\mathbf{v}$ is the vector of node voltages and $\mathbf{i}$ is a vector of the independent source currents [54].

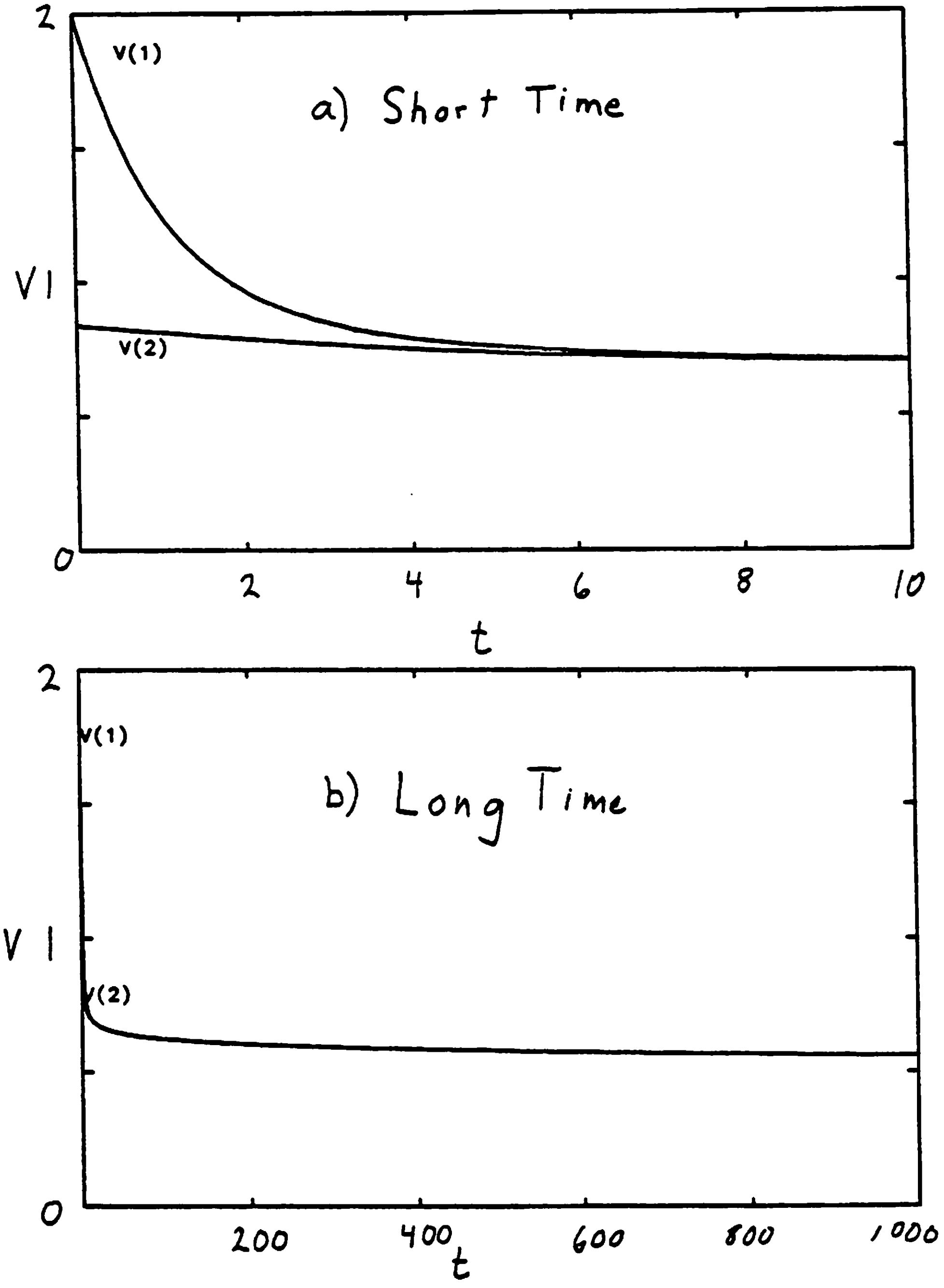


Figure 3.3 : Example Circuit DCTRAN Waveforms

The branch relations for the three branches for this example are

$$I_D = I_s(e^{V_2/V_t} - 1) \tag{3.1}$$

$$I_R = \frac{V_1 - V_2}{R} \tag{3.2}$$

$$I_C = C\,\frac{dV_1}{dt} \tag{3.3}$$

where $V_t = \frac{kT}{q} \approx 26mV$ at $300K$. Kirchoff's Current Laws for the two nodes are

$$I_C + I_R = 0 \tag{3.4}$$

$$I_R - I_D = 0 \tag{3.5}$$

Combining these equations gives

$$C\,\frac{dV_1}{dt} + \frac{V_1 - V_2}{R} = 0 \tag{3.6}$$

$$\frac{V_1 - V_2}{R} - I_s(e^{V_2/V_t} - 1) = 0 \tag{3.7}$$

Equations 3.6 and 3.7 are a system of nonlinear ordinary differential equations. Before going on to solve this system of equations, the dc case where there is no time dependence is first examined.

A good treatment of the solution of systems of nonlinear equations using iterative methods is given in Chapter 7 of [55] and the application of these techniques to circuit analysis problems is given in [52]. At this point the system of equations will be reduced to one equation in order to contrast different solution methods.

The dc equivalent of this example is shown in Figure 3.4. Here the capacitor is replaced with a voltage source in order to reduce the problem to the solution of only the voltage on node 2. Equation 3.6 can now be replaced with

$$V_1 = 2 \tag{3.8}$$

Equation 3.7 is now one nonlinear equation in one unknown. All iterative methods for solving nonlinear equations involve a scheme for finding an equation for the variable in terms of itself. In general, these are called *fixed-point* methods since at convergence the solution is stationary. Rearranging Equation 3.7 in the simplest way to solve for the single unknown, V_2:

$$V_2 = V_1 - R\ I_s\ (e^{V_2/V_t} - 1) \tag{3.9}$$

Using values of $I_s = 10^{-14}$ and $V_t = 26mV$, if m is the iteration number index, then using a starting guess for V_2 of 0 Volts results in the following sequence.

m	0	1	2	3	4	5
V_2	0	2.0	-2×10^{19}	2.0	-2×10^{19}	2.0

Obviously, the sequence oscillates quite quickly for this formulation. Figure 3.5 shows the linearized circuit equivalent for this equation where the diode is being modeled with a current source with a value given by Equation 3.1. This corresponds to a flat load line as shown in Figure 3.6.

Equation 3.9 could be rewritten by rearranging and taking the logarithm of both sides to give

$$V_2 = V_t\ \ln(\frac{V_1 - V_2}{R\ I_s} + 1) \tag{3.10}$$

This corresponds to solving for the branch voltage across the diode given the current through it. Equation 3.10 converges extremely rapidly as seen in the following table.

m	0	1	2	3	4
V_2	0	0.85616	0.84164	0.84196	0.84196

Nodal analysis is commonly used in the automatic generation of the circuit equations since it is easy to use element templates when loading the matrix. The nodal formulation of Equation 3.9 can be improved by using a first-order (Norton) model for the diode as shown in Figure 3.7 which corresponds to the load line in Figure 3.8. This results in a Newton-Raphson iteration. The nodal form of the equations can be written down directly from the circuit.

$$\frac{V_2 - V_1}{R} + V_2 G_d + G_N = 0 \tag{3.11}$$

where

$$G_d = \frac{dI_d}{dV_2} \tag{3.12}$$

and

$$G_N = I_d - G_d V_2 \tag{3.13}$$

Substituting Equations 3.12 and 3.13 into Equation 3.11 and rearranging Equation 3.11 to produce V_2 in terms of itself gives

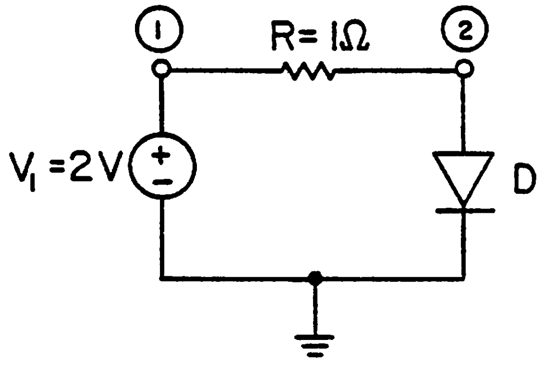


Figure 3.4 : DC Example

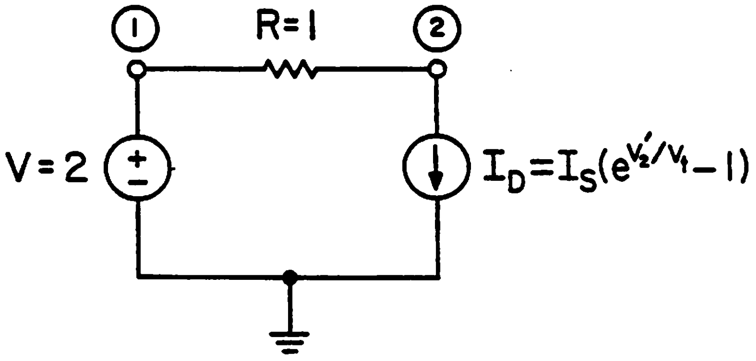


Figure 3.5 : Constant Source Approximation for Diode

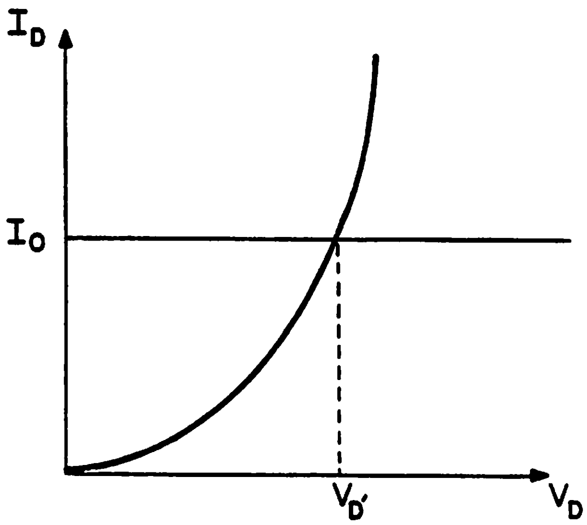


Figure 3.6 : Zero Order Load Line

$$V_2 = \frac{Is + \frac{V_1}{R} - Ise^{V'_2/V_t}(1 - \frac{V'_2}{V_t})}{\frac{1}{R} + \frac{Is}{V_t}e^{V'_2/V_t}} \tag{3.13}$$

The iteration now converges as shown in the following table and in Figure 3.9.

m	0	1	2	3	5	10	20	40	50
V_2	0	2.0	1.974	1.948	1.896	1.766	1.506	0.986	0.84196

To summarize the points about nonlinear equations, in general, a fixed-point iteration is not guaranteed to converge. "Zeroth-order" models for nonlinear devices may be made to converge but "first-order" models which result in a Newton-Raphson iteration converge more quickly. Higher order models in principle could be used, but this results in a system of polynomial equations which itself is not linear and must be solved. Even though it can be shown

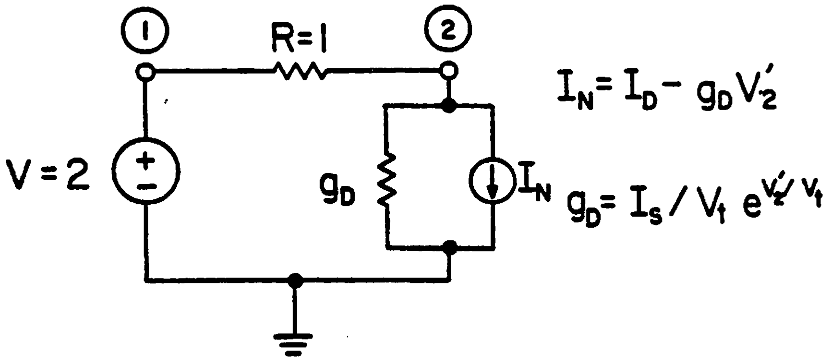


Figure 3.7 : First Order Approximation for Diode

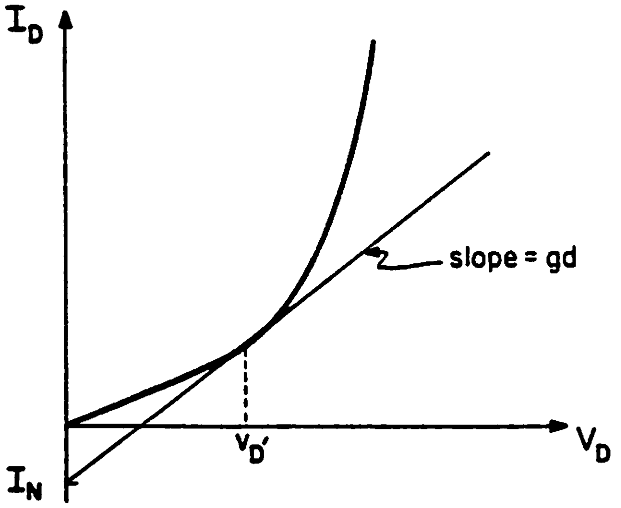


Figure 3.8 : First Order Load Line

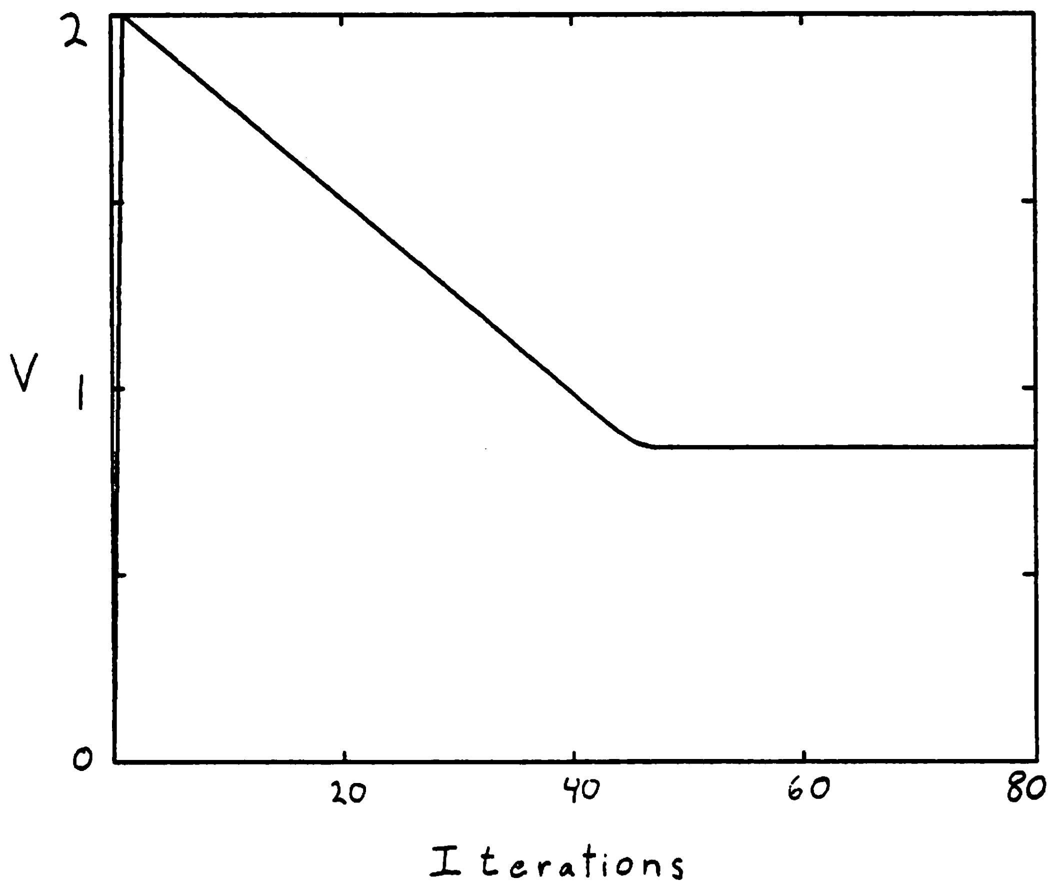


Figure 3.9 : Convergence Rate of Newton Raphson

that a Newton-Raphson iteration converges quadratically near a point of contraction, Figure 3.9 demonstrates that the Newton-Raphson iteration may not converge quadratically in regions which are far from the solution.

This example can be understood by analogy with feedback amplifiers if the iteration number is treated as a discrete time and the sequence of voltage iterates is a transient waveform. Using this view, one can see that the current source and conductance of the linearized model are no longer independent and now depend on the previous iterate value of the voltage. The current source becomes a controlled current source and the conductance becomes a controlled conductance. The oscillation of the iteration when the diode is modeled only by a (constant) current source is now understood to result from the gain of the controlled current source coupled with the controlling voltage time lag from the previous iteration. The conductance in the first order model serves to add local feedback to stabilize the gain in much the same way as emitter degeneration is used in emitter followers to reduce the gain. This type of circuit analogy may prove useful in finding ways of improving simulation algorithms and is a topic for further study.

3.2.2. Transient Solution

The focus of the previous section was on solving a set of nonlinear equations by various ways of linearizing and iterating. The next step is to make use of these techniques to solve the original differential equation as defined in Equations 3.6 and 3.7

$$C\,\frac{dV_1}{dt} + \frac{V_1 - V_2}{R} = 0 \tag{3.6}$$

$$\frac{V_1 - V_2}{R} - I_s(e^{V_2/V_t} - 1) = 0 \tag{3.7}$$

The most common technique used for circuit simulation is the method of finite differences.

Basically, *differential* equations become *difference* equations by noting that

$$\frac{dV}{dt} \approx \frac{\Delta V}{\Delta t}. \tag{3.14}$$

The smaller Δt, the better the approximation becomes. All of the various integration methods correspond to different ways of making this approximation to the derivative. Figure 3.10 shows an arbitrary waveform and a possible choice for the discrete time-points at which solutions are computed. If the solution about to be computed is V_3 as shown, and the derivative is approximated by

$$\frac{dV_3}{dt} \approx \frac{V_2 - V_1}{t_2 - t_1} \tag{3.15}$$

then the integration method is called Forward Euler (FE). Forward Euler is also said to be an *explicit* method since the estimate of the derivative can be computed explicitly prior to beginning the calculation of V_3. Explicit methods suffer from stability problems. The simplest *implicit* method is called Backward Euler (BE) and corresponds to the approximation

$$\frac{dV_3}{dt} \approx \frac{V_3 - V_2}{t_3 - t_2}. \tag{3.16}$$

Note that since the right hand side of Equation 3.16 references V_3, the derivative estimate is linear in the unknown variable. The Trapezoidal integration method is the most accurate of the second order methods [54] and may be viewed as the average of Forward and Backward Euler:

$$\frac{dV_3}{dt} \approx \frac{1}{2}\left[\frac{V_3 - V_2}{t_3 - t_2} + \frac{V_2 - V_1}{t_2 - t_1}\right] \tag{3.17}$$

These approximate derivatives are then used in branch equations for elements such as capacitors:

$$I_C = C\,\frac{dV}{dt}. \tag{3.18}$$

In order to build on the nonlinear equation solving techniques developed in the previous section, the capacitor must be replaced by some collection of dc elements which, when used in a dc calculation, will result in the proper voltage at this time. For example, since all of the terms in the Forward Euler approximation are independent of the unknown variable, the

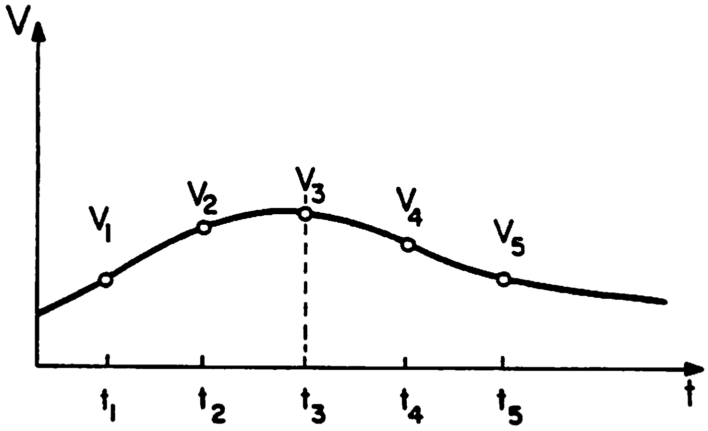


Figure 3.10 : Discretizing Time

capacitor can be replaced by a current source of constant value

$$I = C \frac{V_2 - V_1}{t_2 - t_1}. \tag{3.19}$$

Similarly, since Backward Euler is linear in the unknown voltage, the capacitance can be replaced by a parallel combination of a constant current source and a conductance with values

$$\begin{aligned} I &= \frac{-C}{t_3 - t_2} V_2 \\ G &= \frac{C}{t_3 - t_2}. \end{aligned} \tag{3.20}$$

Trapezoidal integration likewise results in a current source and a conductance but with values

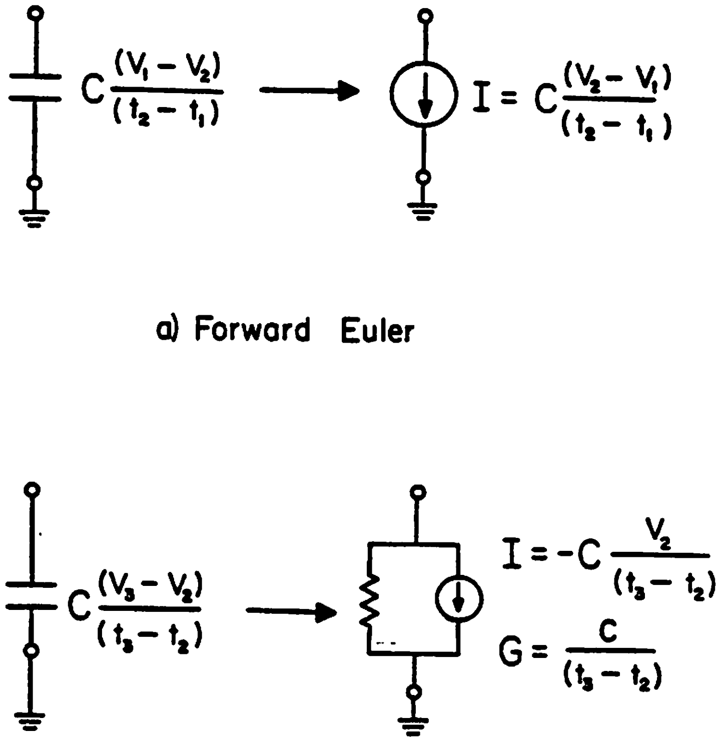
$$\begin{aligned} I &= \frac{-C}{2t_3 - t_2} V_2 + \frac{C}{2t_2 - t_1} V_2 \\ G &= \frac{C}{2t_3 - t_2}. \end{aligned} \tag{3.21}$$

Figure 3.11 illustrates the capacitor companion models using Forward and Backward Euler integration.

Some way of choosing where to place solution points (or what time-steps to use) is needed in order to complete an algorithm for solving differential equations. A discussion of time-step and error control will be deferred to Section 3.4 where error measures will be examined more fully.

3.3. Summary of Existing Techniques

The important algorithmic choices made in various circuit simulators are highlighted in this section. For more detailed information on these programs, for SPICE2 see [4], for MOTIS-C see [9], for SPLICE1 see [1] and [3], for RELAX see [53], and for SAMSON see [13]. For a current introduction to computer aided circuit analysis techniques see [52]. A broad survey of decomposition techniques for the simulation of large-scale integrated circuits can be found in [56]. For a review of timing analysis methods and relaxation techniques applied to circuit simulation see [57].


$C\frac{(V_1 - V_2)}{(t_2 - t_1)}$
$I = C\frac{(V_2 - V_1)}{(t_2 - t_1)}$
$C\frac{(V_3 - V_2)}{(t_3 - t_2)}$
$I = -C\frac{V_2}{(t_3 - t_2)}$
$G = \frac{C}{(t_3 - t_2)}$

a) Forward Euler

b) Backward Euler

Figure 3.11 : Capacitor Companion Models

The differences among the simulators lie mainly in how the elementary techniques described in the preceding sections are combined. In particular, one point of departure is how the systems of nonlinear equations are solved. In timing simulators such as MOTIS and the original SPLICE1, only one iteration of the linearized equations is performed. To achieve accuracy, the time-step is made small enough that the grounded capacitor that is required at every node effectively decouples the equations and the coupling between them is performed through time. In conventional circuit simulators such as SPICE2, the nonlinear equations are linearized, the linear equations are solved using LU factorization, and the procedure is iterated until convergence is attained. Waveform relaxation is used in RELAX, where a decoupled analysis is performed on waveforms which are solved for the whole simulation period and then relaxed together. Iterated timing analysis (ITA) is a new technique [5] in which the nonlinear equations are solved by a decoupled analysis where each equation is linearized and solved once and the system of nonlinear equations is iterated to convergence. ITA is described in Chapter 4.

3.4. Error Measures

Error measures are used by the simulator to judge the quality of the the computed results. Specifying the limits on these error measures is the primary way that the user has of indicating what accuracy is needed. The ways of expressing error constraints need to be sufficiently flexible in order to prevent excessive time spent calculating results which are more accurate than needed. Further, an increased understanding of convergence criteria can improve the reliability and quality of the results. In the next section, methods for specifying error tolerances are described. An extension of the backward difference formula (BDF) method of Brayton, *et al* [58] is presented in the following section which extends the idea of truncation error to take into account time tolerances.

3.4.1. Specifying DC Error

The objective of error specifications is to express the user's desired level of accuracy for the computed solution. This error specification is translated by the simulation program into a criterion for when to terminate the iteration of the nonlinear system of equations. The goal is to perform just the necessary amount of computation to guarantee the answer to the degree of accuracy required by the user. This may or may not be achieved, depending on the ability of the user to express the characteristics that are important, and the ability of the simulation algorithm to solve the problem circuit.

Most circuits simulator provide three variables for specifying accuracy: absolute voltage tolerance, absolute current tolerance and relative tolerance. Relative tolerance indicates the acceptable error as a fraction of the correct solution. The purpose of the absolute tolerances is to prevent excessive computation of a result that tends toward zero, since a relative error tolerance goes to zero as the result goes to zero. Since the relative tolerance indicates the number of significant figures for the calculations, it should be larger than the minimum resolution of the machine so that numerical noise from roundoff errors does not cause excess computation. The tolerance used in calculations is then the maximum of the relative tolerance times the variable and the absolute tolerance for that variable. Convergence occurs when the error is less than the tolerance. The error is usually approximated by the change in voltage and current from one iteration to the next. If the iteration-count is considered as a discrete time variable, then the iterative process can be viewed in a way similar to the transient problem. The convergence criterion then translates into a condition that the "convergence waveform" is *flat enough*. Figure 3.12 shows that the criterion can be visualized as a requirement that the convergence waveform enter and leave the box delimited by a vertical distance equal to the tolerance and a horizontal distance equal to one iteration. This criterion also can be stated as the condition that the locus of points on the convergence waveform in the interval from the preceding iteration to the current iteration all lie inside the closed n-ball with radius equal to the max norm of the tolerances. False convergence can occur when the convergence waveform is flat enough to satisfy the convergence criterion but still has not attained the desired accuracy. Proofs about the rate of convergence of algorithms are restricted to asymptotic results in open n-balls about the point of contraction [55]. One way of decreasing the likelihood of false convergence when far from the point of contraction is to require the convergence tolerances to be satisfied for more than just one iteration. This corresponds to increasing the length of the box in Figure 3.12 holding the other dimensions constant.

The usual method of halting iteration when the per-iteration change in the solution variable drops below the tolerance implies that the per-iteration change is a good estimate of the remaining error from the true solution. This may not be correct if the computed solution

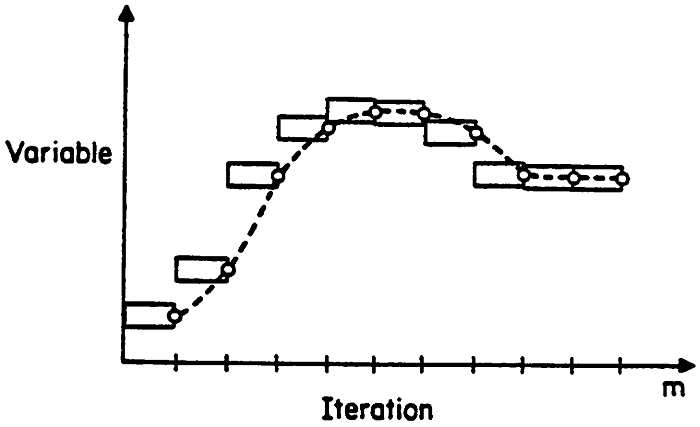


Figure 3.12 : Convergence Criterion

is not close enough to the answer to be in the asymptotic convergence region or if the asymptotic rate of convergence is low. Figure 3.9 has already demonstrated that even though the Newton-Raphson method can be shown to be quadratically convergent in some non-zero interval around the solution, the convergence rate can be much less than quadratic when far from the solution. Asymptotic convergence rates are defined in terms of the remaining error from the true solution and, for quadratic convergence, can be written:

$$\epsilon_{j+1} \leqslant k \ \epsilon_j^{\ 2} \tag{3.22}$$

where ϵ_j, ϵ_{j+1}, ϵ_{j+2} are the errors (difference between the computed solution and the true solution) at iterations j, $j+1$ and $j+2$, respectively. A linear convergence rate is written:

$$\epsilon_{j+1} \leqslant k \ \epsilon_j \ . \tag{3.23}$$

Figure 3.13 shows the sequence of diode voltage and current iterates just before convergence for the resistor-diode example of Figure 3.9. The diode current values decrease linearly (not quadratically) by a factor of 2.718 per iteration (which corresponds to a k of .37) until a region close to the final answer where the current values decrease quadratically. The voltage values decrease by a constant amount per iteration which can be shown to be $\Delta V = V_t \ln(k) = .026V$ due to the logarithmic relationship between voltage and current for diodes. This observation demonstrates that it is not sufficient to use only voltage or current as the variable for convergence tests when iterating using nonlinear devices. Further, the per-iteration change in the variable can be a very bad approximation to the remaining error in the computed solution. Under different circuit conditions and looser error tolerances, the sequence of voltages shown in Figure 3.13 might be considered stationary even though it is not approaching an asymptote.

A better estimate of the distance remaining to the asymptotic value can be made by saving information about previous iterations. Figure 3.14 illustrates a linearly convergent sequence of iterates that converges with a constant $k = \frac{1}{2}$ to a value of zero. The values d_1 and d_2 are defined to be: $d_1 = \epsilon_j - \epsilon_{j+1}$ and $d_2 = \epsilon_{j+1} - \epsilon_{j+2}$. If d_1 and d_2 are measured during iteration, an estimate of ϵ_j can be made using the equality condition in Equations 3.22 and 3.23 which is the worst case:

$$\begin{aligned} d_1 &= \epsilon_j - \epsilon_{j+1} \\ &= \epsilon_j - k\,\epsilon_j = (1-k)\epsilon_j \end{aligned} \tag{3.24}$$

iter	voltage	current
36	1.09000	1.61055×10^{4}
37	1.06400	5.92523×10^{3}
38	1.03800	2.18012×10^{3}
39	1.01201	8.02387×10^{2}
40	0.98605	2.95555×10^{2}
41	0.96014	1.09111×10^{2}
42	0.93439	4.05339×10
43	0.90909	1.53183×10
44	0.88498	6.06080
45	0.86386	2.68934
46	0.84899	1.51783
47	0.84281	1.19683
48	0.84196	1.15865
49	0.84195	1.15804
50	0.84195	1.15804

Figure 3.13 : R-Diode Convergence Values

$$d_2 = \epsilon_{j+1} - \epsilon_{j+2} \tag{3.25}$$
$$= k\ \epsilon_j - k^2 \epsilon_j = k\ (1-k)\epsilon_j$$

so that

$$\frac{d_2}{d_1} = k \tag{3.26}$$

and

$$\epsilon_j = \frac{d_1}{1-k} = \frac{d_1^2}{d_1 - d_2}. \tag{3.27}$$

Equation 3.27 indicates that using the per-iteration change in the variable as an estimate of the distance to the asymptotic solution can be arbitrarily bad as k approaches 1. For a value of $k = \frac{1}{2}$, d_1 is only half of the error distance to the asymptote.

Equation 3.27 can be applied to the data of Figure 3.13 as an example. Using iteration 36 as j, $d_1 = 1.02\times10^4$ and $d_2 = 3.74\times10^3$ for the current variable. This gives $k = .368$ and $\epsilon_j = 1.610119\times10^4$. Clearly ϵ_j is larger than d_1. Further, ϵ_j can be used to predict the asymptote to be 1.19 which is not far from the true answer of 1.15804. This prediction capability might lead to techniques to reduce the total number of iterations to achieve the solution and is worth further investigation. The use of Equation 3.27 with the voltage variable yields an infinite estimate of ϵ_j since the change in voltage is constant and results in $k = 1$. This is an important result since it provides a simple way of detecting false convergence even when the change per iteration satisfies the simple tolerance condition. Equation 3.27 can be used as part of a more reliable convergence criterion.

The same estimate of error can be made with quadratically convergent sequences but with a slightly more complicated answer. Using d_1 and d_2 as defined above, we have using Equation 3.22:

$$d_1 = \epsilon_j - \epsilon_{j+1} = \epsilon_j - k\ \epsilon_j^2 \tag{3.28}$$

and

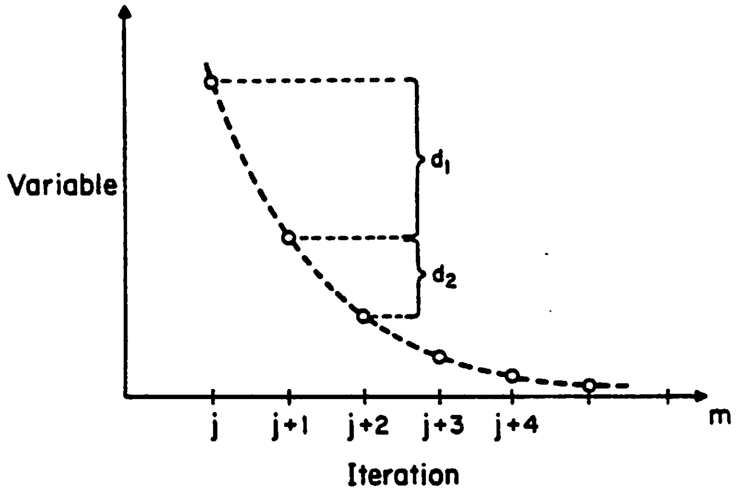


Figure 3.14 : Linearly Convergent Sequence

$$d_2 = \epsilon_{j+1} - \epsilon_{j+2} = \epsilon_{j+1} - k\ \epsilon_{j+1}^2 = k\ \epsilon_j^2 - k^3 \epsilon_j^4. \tag{3.29}$$

Solving these equations for ϵ_j gives:

$$\epsilon_j = \frac{3d_1^2 \pm d_1^{\frac{3}{2}} \sqrt{(4d_2 + d_1)}}{2d_2 - 4d_1} \tag{3.30}$$

which can be seen to have a form similar to that of Equation 3.27.

3.4.2. Specifying Waveform Error

The elements in the network that have *memory*, such as capacitors, are approximated at each solution time point by equivalent linear companion models. These models are chosen to be *consistent* so that as the time-steps approach zero, the approximation becomes exact. The error incurred during each solution that is caused by using a finite time-step is called local truncation error (LTE) [52]. The LTE is estimated by comparing a higher-order method with the one actually used. One way of estimating the higher-order derivatives is the divided differences technique [4]. A better technique is the backward differentiation formula (BDF) method of Brayton *et. al.* [58] which uses the difference between a prediction step and the converged value as an estimate of LTE.

By using a particular order of integration method, the waveform is being approximated with a polynomial of corresponding degree. This implies that when values are needed for the waveform at points between those that were computed, an interpolating polynomial of the proper order is needed. For example, Backward-Euler integration (BE) is a first order method for which a linear interpolation polynomial is appropriate. The polynomials used in BE integration for prediction-correction are illustrated in Figure 3.15. The line segments between calculated points are straight. The difference between the predicted value and the final value after iteration is a measure of the local truncation error made with this time-step. If equal sized time-steps are being taken, then the LTE estimate is one half of the difference between the predicted and corrected values [58].

One problem with this method of time step size control is that if the solution waveform changes rapidly, then the step size may be chosen extremely small in order to follow the waveform within the specified tolerance. The user of the simulator may not care to get that much accuracy for the time behavior of the waveform, but has no way to specify that to the simulator. This problem is illustrated in Figure 3.16 which shows that a small error in time between two waveforms may result in large errors in the waveform variable (voltage in this

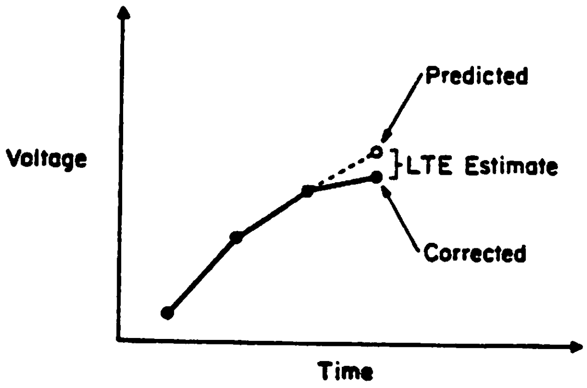


Figure 3.15 : Backward-Euler Predictor-Corrector

case). The BDF method for estimating LTE has been extended to allow the estimation of the error in the time variable in the SPLICE2 program. Figure 3.16 suggests that the condition for acceptance of a time step is that the distance between the computed *waveform* and the true *waveform* is less than the user supplied tolerances. Thus a new measure, the *waveform local truncation error* is made up of both the usual LTE and the local truncation error estimate for time. The distance measure used here is the max norm rather than the Euclidean norm.

Specifically, for Backward-Euler integration of voltage, let $n + 1$ be the next solution point where the previous solutions (v_n, t_n) and (v_{n-1}, t_{n-1}) are taken to be on the true waveform. The relationships between the quantities in these calculations are illustrated in Figure 3.17. The LTE for voltage is:

$$LTE_v = \frac{t_{n+1} - t_n}{t_{n+1} - t_{n-1}} (v_{n+1} - v_{n+1}^{pred}) \tag{3.31}$$

and the LTE for time is then:

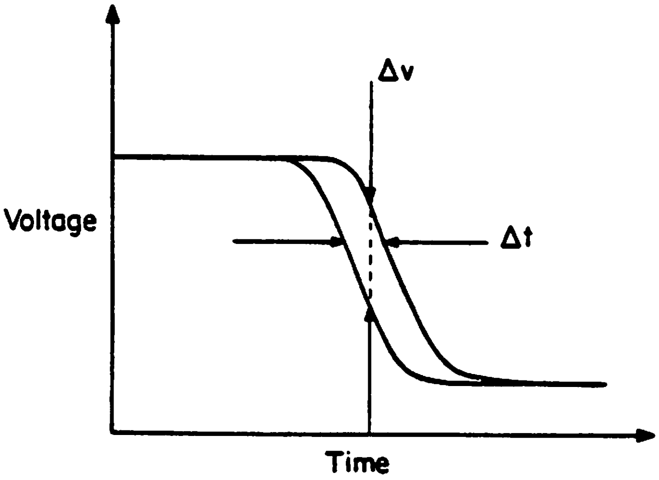


Figure 3.16 : Waveform Error

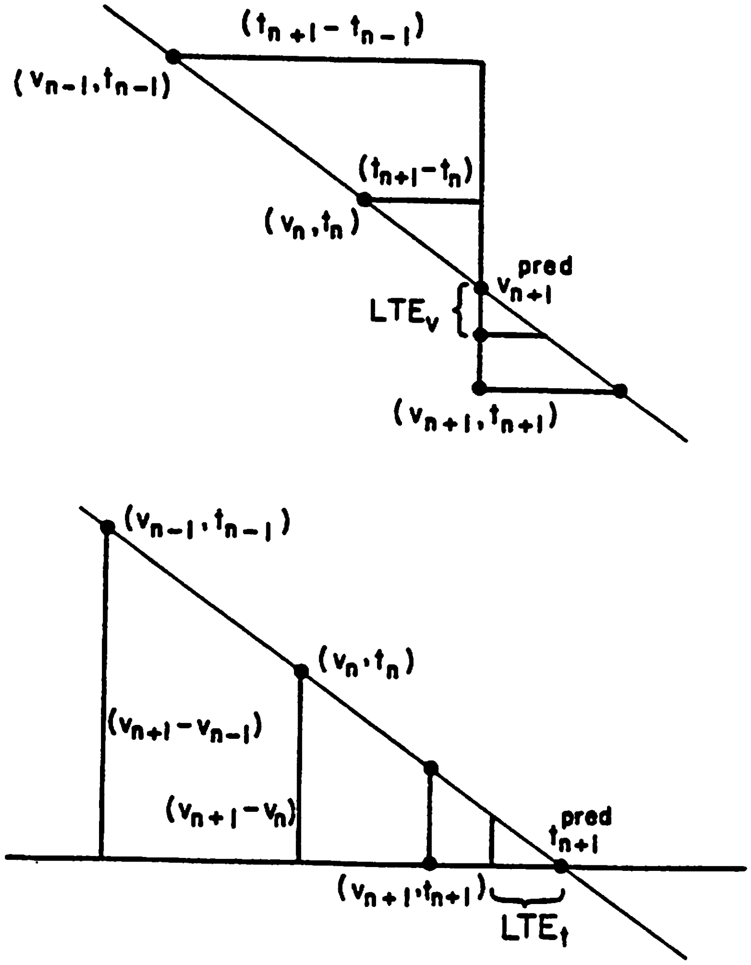


Figure 3.17 : Waveform Local Truncation Error

$$LTE_t = \frac{v_{n+1} - v_n}{v_{n+1} - v_{n-1}} (t_{n+1} - t_{n+1}^{pred}). \tag{3.32}$$

A given pair of voltage and time tolerances corresponds to a slew rate above which the time tolerance will be the constraining criterion and below which, the voltage tolerance will be the constraining criterion.

The waveform local truncation error criterion has been implemented in the SPLICE2 program. It has been found that the addition of the time tolerance reduces the number of solution rejections significantly. Setting the time tolerance to zero reverts to the old method of using only local truncation error on the dependent variable. An example showing the effect of using a time tolerance is summarized in Figure 3.18 and the waveforms are compared in Figure 3.19. This example is a chain of 7 n-channel MOS depletion load inverters with each output loaded by a 0.01pF capacitor to ground except for the fourth output which is loaded only by 0.0001pF. The CPU time for simulation is reduced by 10% and the number of solutions rejected due to truncation error is reduced by 40%. The waveforms shown in Figure 3.19 all differ in time by much less than the 0.5ns that was specified.

3.4.3. Optimal Choice of Time-step

Using the error measures introduced above, the problem specification is reduced to that of finding a solution waveform that is everywhere close enough to the true waveform. It is then possible to define the optimum choice for the placement in time of the solution points. First, to define *close enough* it is useful to introduce a *distance waveform* which is computed as follows. Let $f_1(x,t)$ and $f_2(x,t)$ be two waveforms in $\mathbb{R}^{n \times n}$ which are single-valued on t i.e., not relations, and have the same range of t. The *distance waveform*, $d(x,t)$, between $f_1(x,t)$ and $f_2(x,t)$ is computed:

$$d(x,t) = \min(\min(\|f_1(x,t) - f_2(x,\bullet)\|), \min(\|f_1(x,\bullet) - f_2(x,t)\|)) \tag{3.33}$$

where the norm used is a max norm. Note that the use of the norm implies that time has been scaled so that a unit distance in time is the same as a unit distance in x. The error criterion can now be written compactly as:

Variable	Time Tolerance	
	0ns	0.5ns
CPU Time	91.4	82.4
Rejected Solutions	154	93
Total Iterations	22824	20319
% Useful Iterations	85.3	89.9
Number Windows Used	325	270

Figure 3.18 : Example Using Time Tolerance

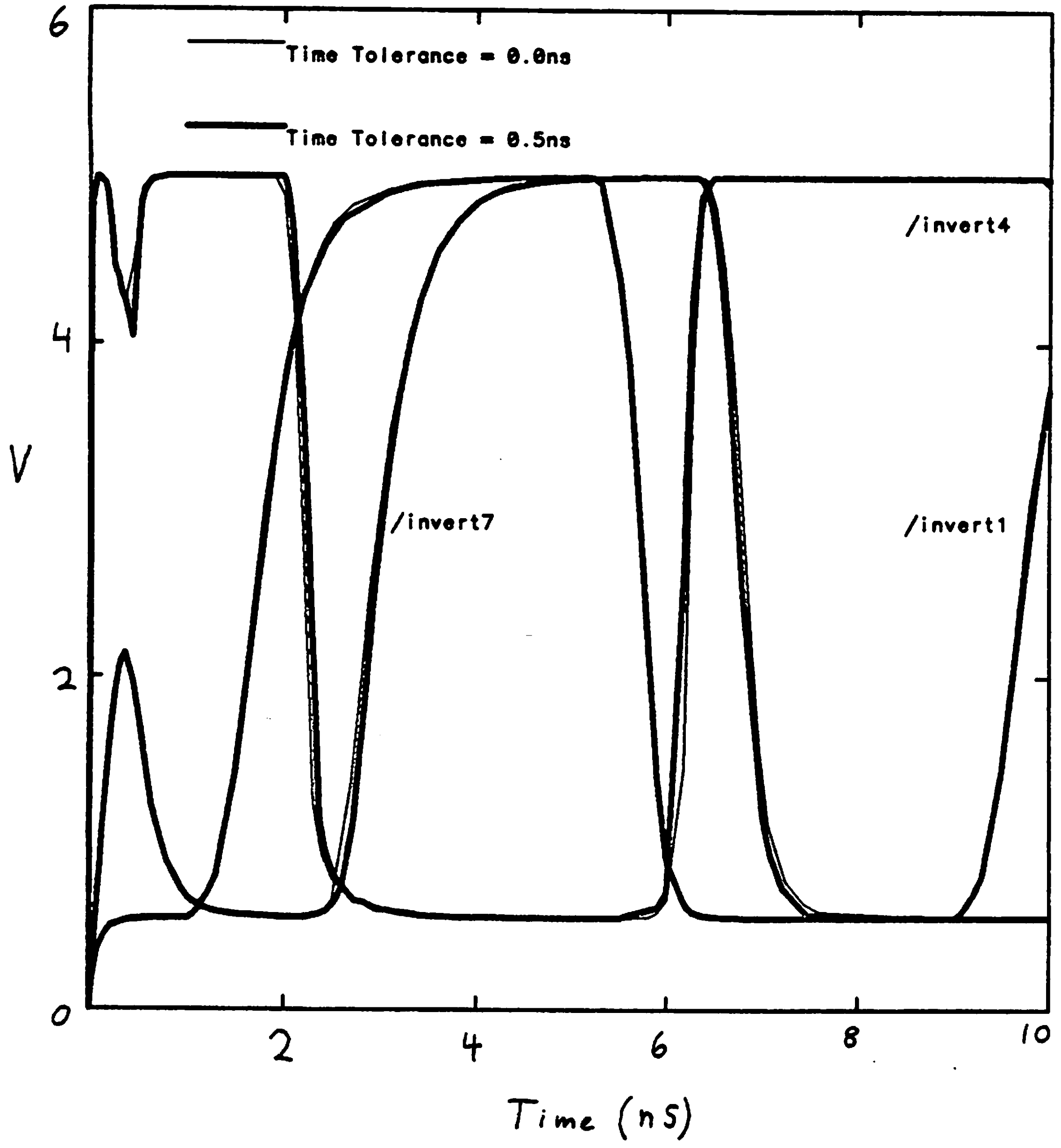


Figure 3.19 : Waveforms for Example

$d(x, t) \leqslant \text{tolerance}, \forall\ t$. (3.34)

Figure 3.20 shows two pairs of example waveforms and the simple difference waveform along with the distance waveform. Note in Figure 3.20a that a slight time delay leads to a very large simple difference but the distance waveform remains fairly constant. Figure 3.20b demonstrates the *filtering* effect of the time tolerance and that glitch pulses tend to be smoothed out.

For a given waveform accuracy, integration method, and corresponding interpolation polynomial, the optimum number of solution points can be defined as the minimum number necessary such that the locus of points described by the solutions and the line segments which are created from them using the interpolation polynomial all lie within the specified waveform accuracy of the true waveform. An example using a linear interpolation polynomial is shown in Figure 3.21.

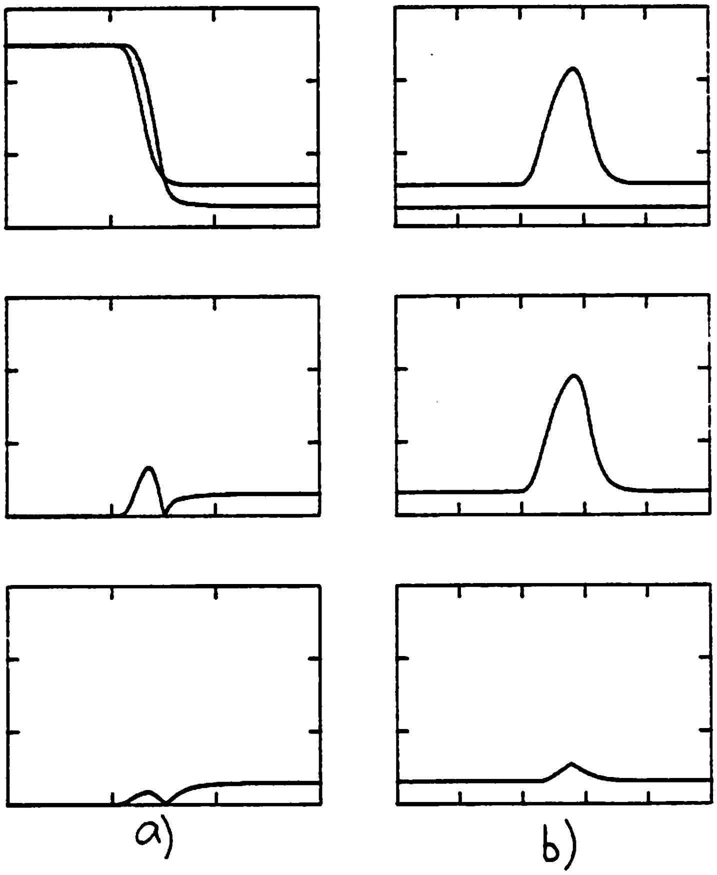


Figure 3.20 : Waveform Distance Examples

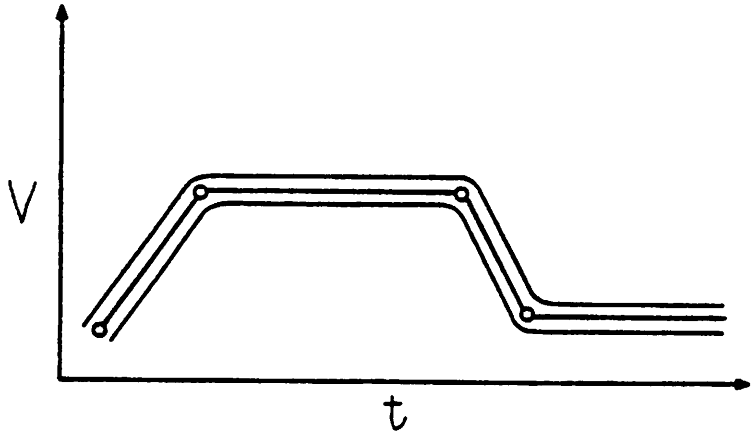


Figure 3.21 : Optimum Time-Steps

3.5. Speedup Factors

There are several properties of circuits that are used to speed the computation of the solution waveforms [13]. The first property is that electrical networks are *spatially sparse*. Spatial sparsity means that there tend to be only a few electrical elements connected to any given wire so that the matrices representing the circuit equations are sparse. Conventional circuit simulators take advantage of this fact by using sparse matrix techniques [59]. Another property of electrical networks which perform logic functions is that they tend to be *temporally latent*. Temporal latency occurs in logic networks because the time required for a logic state change is small compared to the total clock time. This is illustrated in Figure 3.22 in which the upper waveform is a logic signal and the lower waveform is an abstraction of the logic waveform that is a logical one while the waveform is switching and logical zero otherwise. Conventional circuit simulators take advantage of temporal latency by employing a variable time-step, as mentioned previously, to perform computation mainly when the signal is changing. Another property of logic networks is *spatial latency*. Spatial latency is due to the fact that only a small fraction of all signals switch at any given instant of time and represents the non-coherence of the signals. Conventional circuit simulators calculate all of the signals at every time point chosen for solution and thus are not able to take advantage of this latency. The solutions of the signals must be decoupled in order to exploit this property. Temporal latency and spatial latency are usually considered using the term latency for both. Logic simulators and electrical simulators which allow decoupled analysis [1, 3, 13, 29] can exploit the latency in the electrical analysis of logic networks to the fullest.

Decoupled analysis of electrical networks requires some form of time-step control to choose when each signal is computed. If a first-order integration method, such as Backward-Euler, and a corresponding first-order interpolation polynomial are used, then the spacing between solutions for each signal will be proportional to the second derivative of the signal. If the spacing between solutions is τ and the waveform is $f\ (t\ )$ then $\tau \propto \frac{d^2 f\ (t\ )}{dt^2}$. There is a different τ for every signal in the network which results in a distribution of time-steps for the network that varies with simulated time. The ability of the time-step control algorithm to allow each solution to select its own optimum solution points while still computing the correct solution is the key to computational efficiency. When solving for one signal at a time point that depends on another signal which happens to be latent at this time point, the latent

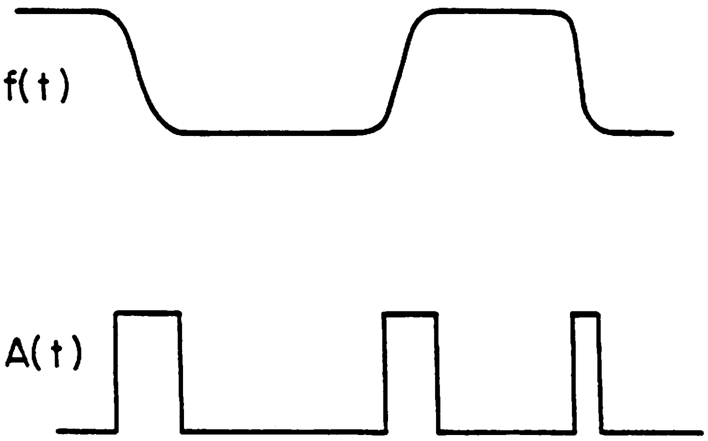


Figure 3.22 : Temporal Latency in Logic

signal is extrapolated or interpolated to determine its present value. If the latent signal is subsequently solved and found to be significantly different from what was assumed at the previous time, then it may be necessary to reject previously accepted solutions. This corresponds to backing up simulated time and re-solving for at least a portion of the electrical network. Thus, there must be some mechanism for recalling data values from the output buffer and replacing them with new values. This can require a large amount of data to be retrieved, as will be shown shortly, unless some limit is placed on the maximum number of points the simulator is permitted to back up. This corresponds to a maximum ratio between the longest time-step and the shortest time-step being taken at any instant of simulated time. What follows is a description of the *windowing* mechanism for buffering and time-step control that has been implemented in the SPLICE2 program.

The windowing mechanism described here uses an in-memory buffer to hold solution points before writing them to the disk mass storage device. This buffer size limits the ratio of the longest time-step to the shortest time-step allowed at any particular time. The choice of the next time-step for a signal is usually based on the estimate of the truncation error made for the time-step just completed. As such, it may grossly overestimate the actual time-step that will be required to satisfy all of the error criteria that have been described previously. If no limit is placed on the largest time-step, then a fast changing signal that depends on a latent signal with a large time-step could be computed at an arbitrarily large number of time points which must then be rejected when the time-step for the latent signal is rejected. This results in both the waste of those solutions already computed and logistical complexity for retrieving those solutions.

An idealized bimodal distribution of time-steps is shown in Figure 3.23. The fast component has an optimum time-step of τ and a fraction, f, of the total signals have this time-step while the slow component has an optimum time-step of $k\ \tau$ and a fraction $1-f$ of the total signals. For a typical logic network, the fraction of active signals, f, is somewhere between 5-20%. With a given distribution, it is possible to evaluate how much the restriction on the maximum time-step affects the CPU time usage. If the buffer size in use allows a maximum time-step of $B\tau$, as shown in Figure 3.23, and if the solution of each signal is assumed to require the same amount of computer effort, then the minimum amount of work per unit time needed to solve the circuit is:

$$\begin{aligned} \text{best} &= \frac{f}{\tau} + \frac{1-f}{k\tau} \\ &= \frac{1+(k-1)f}{k\tau} \end{aligned} \tag{3.35}$$

and the actual amount required by the windowing method if $k \geqslant B$ is:

$$\begin{aligned} \text{actual} &= \frac{f}{\tau} + \frac{1-f}{B\tau} \\ &= \frac{1+(B-1)f}{B\tau}. \end{aligned} \tag{3.36}$$

The ratio between the actual work and best work is:

$$\frac{\text{actual}}{\text{best}} = \frac{f + \frac{1-f}{B}}{f + \frac{1-f}{k}}. \tag{3.37}$$

If $1 \leqslant k \leqslant B$ then the buffer is large enough that it doesn't constrain the time-step and the ratio is 1, which means no extra work is performed. For a given f, the worst-case behavior is when $k = \infty$ and the ratio becomes simply $1 + \frac{1-f}{fB}$. For $f = 10\%$ and a buffer size $B = 9$, the ratio is 2 and as B is increased, the ratio approaches 1 hyperbolically. For a given fraction of latent signals, the buffer can be chosen so that the computation speed is as close as desired to the optimum at the expense of storage. For a latency of 20%, a buffer size $B = 8$ gives a speed within 1.5 of optimum. When $k < \infty$, the situation is even better than stated. These results suggest that only modest buffering requirements are needed for efficient operation of the method with latencies that occur in practice.

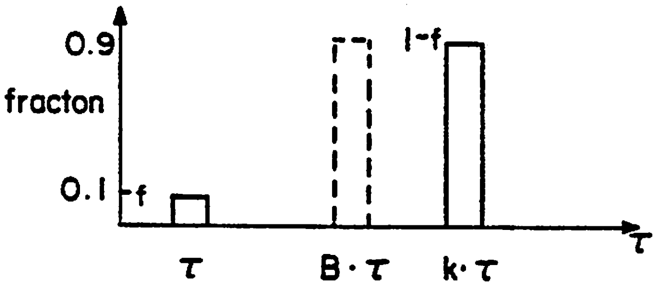


Figure 3.23 : Distribution of Time-Steps

CHAPTER 4

Iterated Timing Analysis

The implementation of the iterated timing analysis (ITA) algorithm in the SPLICE2 program is described in this chapter along with some background information. An excellent review of current research on relaxation-based electrical simulation and a comparison with more conventional approaches is given in [57]. The development in that paper is not repeated here, except for specific points that relate to work reported in this dissertation.

4.1. Problem Formulation

Although ITA can be applied to a wide variety of technologies, it is particularly suited to the analysis of large digital integrated circuits. To help clarify the problem formulation, the following simplifying assumptions are made:

- All resistive elements, including active devices, are characterized by constitutive equations where voltages are the controlling variables and currents are the controlled variables.
- All energy storage elements are two-terminal, possibly nonlinear, voltage-controlled capacitors.
- All independent voltage sources have one terminal connected to ground or can be transformed into independent current sources with the use of the Norton transformation.

Using these assumptions, one can formulate the circuit equations in terms of a nodal analysis that yields N equations in N unknown node voltages [60], where there are $N+1$ nodes in the circuit and node $N+1$ is the reference node, or ground.

The important assumption required by relaxation-based electrical simulators that a two-terminal capacitor be connected from each node of the circuit to the reference node is satisfied easily for circuits that have lumped, parasitic capacitances between circuit interconnect and ground or on the terminals of active circuit elements. The nodal equations then can be written in the form:

$$C(v(t),u(t))\dot{v}(t) = -f(v(t),u(t)),\ 0 \leqslant t \leqslant T \tag{4.1}$$

$$v(0) = V.$$

where $v(t) \in \mathbb{R}^n$ is the vector of node voltages at time t, $\dot{v}(t) \in \mathbb{R}^n$ is the vector of time derivatives of $v(t)$, $u(t) \in \mathbb{R}^n$ is the input vector at time t, $C(\bullet) : \mathbb{R}^n \rightarrow \mathbb{R}^{n \times n}$ represents the nodal capacitance matrix, $f : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}^n$, and:

$$f(v(t),u(t)) = [f_1(v(t),u(t)), f_2(v(t),u(t)), \cdots, f_N(v(t),u(t))]^T \tag{4.2}$$

where $f_i(v(t),u(t))$ is the sum of the currents charging the capacitors connected to node i. Hereafter, Equation (4.2) will be referred to in a simplified form where the time dependencies are expressed implicitly:

$$C(v,u)\dot{v} = -f(v,u) \tag{4.3}$$

The differential equation (Eqn. 4.3) is next converted to a difference equation, as has been illustrated in Chapter 3, by discretizing the term $\dot{v}$ with an appropriate integration method. The terms on the right-hand side of Equation 4.3 can now be moved to the left-hand side and rearranged to give a new vector of nonlinear equations:

$$g(x) = 0\,. \tag{4.4}$$

In conventional circuit simulation, these nonlinear equations are then linearized to give

$$A\ x = b \tag{4.5}$$

which then is solved directly using LU decomposition or Gaussian Elimination. Linearization followed by solution of the equations corresponds to the use of the Newton-Raphson algorithm. The Newton-Raphson method is usually modified using *limiting* algorithms in the nonlinear device models and this results in a *damped* Newton-Raphson algorithm. The linearization/solution process is repeated until the solution converges to within some user specified tolerance of a stationary value.

In the iterated timing analysis (ITA) method, a nonlinear Gauss-Seidel or Gauss-Jacobi relaxation iteration is applied directly to Equation 4.4. The Gauss-Jacobi algorithm can be summarized:

Nonlinear Gauss-Jacobi Algorithm:

```
repeat {
  foreach ( $i$ in $N$ ) {
    solve $g_i(x_1^k, \cdots, x_i^{k+1}, \cdots, x_N^k) = 0$ for $x_i^{k+1}$;
  }
} until ( $\|x^{k+1} - x^k\| \leqslant \epsilon$ )
```

that is, until convergence is obtained. The Gauss-Seidel algorithm can be summarized:

Nonlinear Gauss-Seidel Algorithm:

```
repeat {
  foreach ( $i$ in $N$ ) {
    solve $g_i(x_1^{k+1}, \cdots, x_i^{k+1}, \cdots, x_N^k) = 0$ for $x_i^{k+1}$;
  }
} until ( $\|x^{k+1} - x^k\| \leqslant \epsilon$ ).
```

The difference between the Gauss-Jacobi and Gauss-Seidel algorithms is that with Gauss-Seidel, the new values computed during the iterative process are used immediately whereas with Gauss-Jacobi, the new values are used only after a complete sweep through all of the equations. Relaxing the nonlinear equations directly avoids solving the linear system of equations which, for large networks, can require a substantial amount of time [57].

4.2. Relaxation of Systems of Equations

In the system of nonlinear equations in Equation 4.4, the time variable has been absorbed into constant factors by the application of an integration method. This means that at a given point in time of the transient simulation, the system of equations is equivalent to a system of equations for a purely dc problem. Thus, it is of interest to look more closely at the dc part of the DCTRAN problem. In the rest of this section, the solution of linear and then nonlinear systems of equations by relaxation methods is explored.

4.2.1. Linear Equations

If the problem in Equation 4.4 happens to be linear, then the equations can be rewritten as

$$A\ x = b\ . \tag{4.6}$$

The nonlinear Gauss-Jacobi and Gauss-Seidel algorithms can be applied to the linear case and are expressed in matrix form as follows [61]. Let A be split into $L + D + U$, where $L \in \mathbb{R}^n$ is strictly lower triangular, $D \in \mathbb{R}^n$ is diagonal, and $U \in \mathbb{R}^n$ is strictly upper triangular. Then the two methods have the following form:

Gauss-Jacobi:

$$Dx^{k+1} = -(L + U)x^k + b \tag{4.7a}$$

or

$$x^{k+1} = -D^{-1}((L + U)x^k - b) \equiv M_{GJ}\, x^k + D^{-1}b \tag{4.7b}$$

and

Gauss-Seidel:

$$(L + D)x^{k+1} = -Ux^k + b \tag{4.8a}$$

or

$$x^{k+1} = -(L + D)^{-1}(Ux^k - b) \equiv M_{GS}\, x^k + (L + D)^{-1}b\ , \tag{4.8b}$$

where x^k is the value of x at the k-th iteration and M_{GJ} and M_{GS} are "iteration matrices". A modification to the basic Gauss-Seidel method which can improve the convergence rate dramatically is the *point successive over-relaxation* (SOR) *iterative metl od* [61] which results from using a computed value for the variable that is different from the one calculated during the iteration:

$$\hat{x}^{k+1}{}_i = x^k{}_i + \omega(x^{k+1}{}_i - x^k{}_i) \tag{4.9}$$

and can be written in matrix form:

$$x^{k+1} = (D + \omega L)^{-1}\Big[((1 - \omega)D - \omega U)x^k + \omega b\Big]. \tag{4.10}$$

If the value of ω is equal to one, the method is equivalent to the normal Gauss-Seidel method. A value of ω greater than one corresponds to over-relaxation and a value of ω less than one corresponds to under-relaxation.

In order to compare the convergence properties of these methods, it is useful to summarize here some definitions and theoretical results from the study of matrix iterative techniques [61]. In the following definitions, A is an arbitrary $n \times n$ complex matrix that corresponds to some iterative process

$$\mathbf{x}^k = A^m \mathbf{x}^{k-m} \tag{4.11}$$

where x is a vector in $\mathbb{R}^n$. The iteration matrices M_{GS} and M_{GJ} are typical examples of such a matrix. The *spectral radius* of a matrix is equal to the magnitude of the eigenvalue with the largest magnitude:

$$\rho(A) = \max_{1 \leqslant i \leqslant n} |\lambda_i| \tag{4.12}$$

and the *spectral norm* is

$$||A|| = \sup_{\mathbf{x} \neq 0} \frac{||A\mathbf{x}||}{||\mathbf{x}||} . \tag{4.13}$$

Further,

$$||A|| \geqslant \rho(A) . \tag{4.14}$$

If, for some positive integer m, $||A|| < 1$, then

$$R(A^m) \equiv -\ln[\, ||A^m||^{1/m} \,] = \frac{-\ln||A^m||}{m} \tag{4.15}$$

is the *average rate of convergence for m iterations* of the matrix A. For m sufficiently large,

$$\lim_{m \to \infty} R(A^m) = -\ln \rho(A) \equiv R_\infty(A) \tag{4.16}$$

is the *asymptotic rate of convergence* of A. If A is convergent, then

$$R_\infty(A) \geqslant R(A^m) \tag{4.17}$$

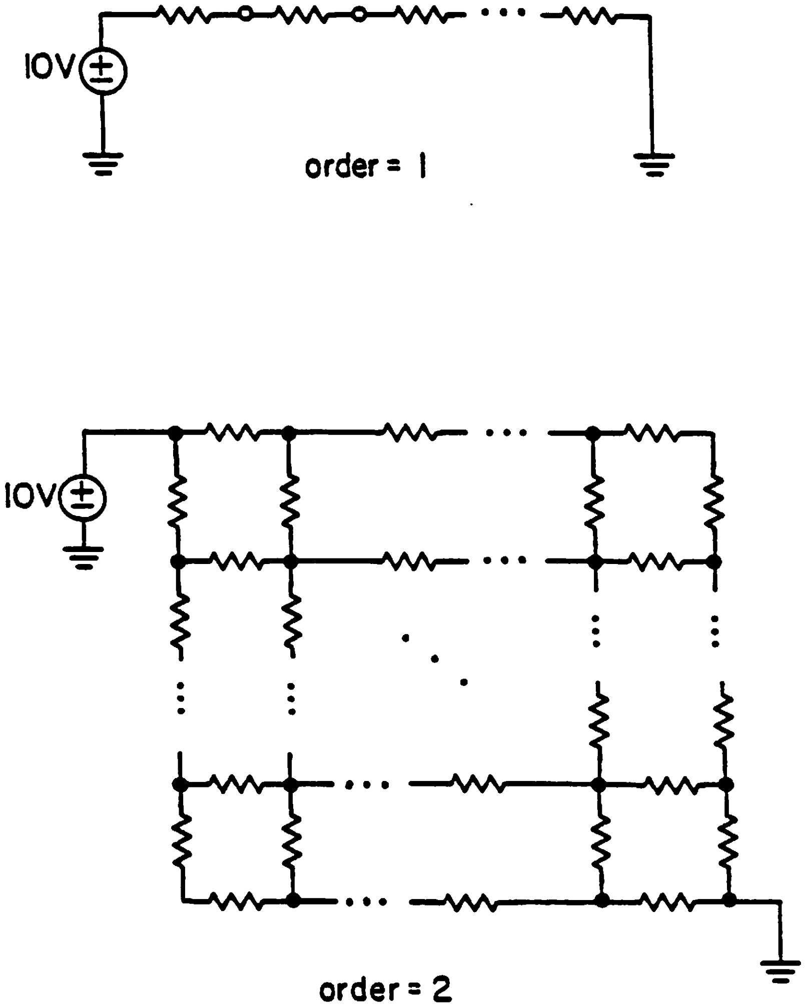
for any positive integer m for which $||A^m|| < 1$.

It is now possible to compare the rates of convergence of the Gauss-Jacobi and Gauss-Seidel iterative methods. The Stein-Rosenberg theorem states that for non-negative matrices, Gauss-Jacobi and Gauss-Seidel methods are either both convergent or both divergent and, if convergent, then the asymptotic rate of convergence of the Gauss-Seidel method is *iteratively faster* than that of the Gauss-Jacobi method. Another advantage of the Gauss-Seidel method of Equation 4.8b is that the new values (x^{k+1}) can be stored over the old values (x^k) as they are computed whereas with the Gauss-Jacobi method of Equation 4.7b, the old values must be saved.

A series of experiments using resistor arrays and Gauss-Seidel iteration of the equations have been performed using the SPLICE2 program to test the theory and develop new ways of speeding convergence. The test networks are resistor arrays of variable order and dimension as is illustrated in Figure 4.1. An order of 1 corresponds to a linear array of resistors and an order of 2 corresponds to a square matrix of resistors. Appendix C contains tables of results on the number of iterations required to solve these networks for arrays of order one through three. Theoretical estimates of the number of iterations (N_∞) needed to solve the equations to a tolerance of 0.001% for the linear array case assuming that the asymptotic rate of convergence applies throughout the iterative process are summarized in Figure 4.2 for both the Gauss-Jacobi iterative method and the Gauss-Seidel iterative method. The estimates of the number of iterations for convergence (N_∞) are based on the asymptotic rate of convergence (R_∞) by noting that $1/R_\infty$ is the number of iterations required for the error to be reduced by a factor of $1/e$. Thus, if the relative tolerance is ϵ and $|\lambda_{\max}|$ is the magnitude of the largest eigenvalue, then the estimate of the number of iterations for convergence within that tolerance is

$$N_\infty = \frac{\ln(\epsilon)}{\ln(|\lambda_{\max}|)} . \tag{4.18}$$

Also shown in Figure 4.2 are the magnitudes of the largest eigenvalues ($|\lambda_{\max}|$) of the corresponding iteration matrices (M_{GJ} and M_{GS}) with a value of $\omega = 1$ and an estimate of the best value of the over-relaxation parameter (ω_b) for use in the Gauss-Seidel SOR iteration. The optimum relaxation parameter (ω_b) is computed using [62]



order = 1

order = 2

Figure 4.1 : Resistor Arrays

Gauss-Seidel						
N	$\lvert\lambda_1\rvert$	$N_{1\infty}$	$\lvert\lambda_b\rvert$	$N_{b\infty}$	ω_b	$\frac{N_{1\infty}}{N_{b\infty}}$
3	0	0	0	0	1	1
4	0.25	5.0	0.07179	2.6	1.0718	1.9
5	0.5	10.0	0.17177	3.9	1.1715	2.5
6	0.65450	16.3	0.25961	5.1	1.2596	3.2
7	0.75000	24.0	0.33333	6.3	1.3333	3.8
8	0.81174	33.1	0.39481	7.4	1.3948	4.5
9	0.85355	43.6	0.44646	8.6	1.4464	5.1
10	0.88302	55.5	0.49029	9.7	1.4902	5.7
11	0.90451	68.8	0.52786	10.8	1.5278	6.4
12	0.92062	83.5	0.56075	11.9	1.5603	7.0
13	0.93301	99.6	0.58879	13.0	1.5887	7.6
14	0.94272	117.1	0.61379	14.2	1.6138	8.3
15	0.95048	136.0	0.63596	15.3	1.6359	8.9
16	0.95677	156.3	0.65575	16.4	1.6557	9.5
17	0.96194	178.0	0.67351	17.5	1.6735	10.2
18	0.96623	201.1	0.68955	18.6	1.6895	10.8
19	0.96984	225.6	0.70409	19.7	1.7041	11.5
20	0.97291	251.5	0.71734	20.8	1.7173	12.1

a) Gauss-Seidel

Gauss-Jacobi		
N	$\lvert\lambda\rvert$	N_{∞}
3	0	0
4	0.5	9.9
5	0.70710	19.9
6	0.80901	32.5
7	0.86602	48.0
8	0.90097	66.2
9	0.92388	87.2
10	0.93969	111.0
11	0.95105	137.7
12	0.95949	167.1
13	0.96592	199.3
14	0.97094	234.3
15	0.97492	272.1
16	0.97814	312.7
17	0.98078	356.1
18	0.98297	402.3
19	0.98480	451.3
20	0.98636	503.1

b) Gauss-Jacobi

Figure 4.2 : Eigenvalues and ω for Linear Array

$$\omega_b = \frac{2}{1 + \sqrt{1 - |\lambda_{\max}|}} \tag{4.19}$$

which is valid for certain classes of matrices. Carre also gives a formula and procedure for estimating the largest eigenvalue and predicting the best relaxation factor during iteration [62]. He also points out that it is important not to exceed the optimum value of ω since violent oscillations of the solution can occur due to complex eigenvalues of the resulting iteration matrix.

The results for the linear arrays of resistors are summarized in Figure 4.3 which shows the theoretical and measured values for the optimum SOR factor along with the iteration reduction ratios. The measured values of ω_b are fairly close to the predicted ones. The speedup ratios are less than predicted primarily due to the difference between the average and asymptotic convergence rates.

Gauss-Seidel				
N	ω_b	ω_b (meas)	$\frac{N_{1\infty}}{N_{b\infty}}$	$\frac{N_{1\infty}}{N_{b\infty}}$ (meas)
3	1	1.05	1	1.0
4	1.0718	1.09	1.9	1.4
5	1.1715	1.20	2.5	1.7
6	1.2596	1.30	3.2	1.8
7	1.3333	1.39	3.8	2.1
8	1.3948	1.4	4.5	2
9	1.4464	1.5	5.1	2.4
10	1.4902	1.52	5.8	2.5
11	1.5278	1.57	6.4	2.5
12	1.5603	1.61	7.0	3.1
13	1.5887	1.62	7.6	2.8
14	1.6138	1.69	8.3	3.1
15	1.6359	1.70	8.9	3.1
16	1.6557	1.70	9.5	3.3
17	1.6735	1.74	10.2	3.4
18	1.6895	1.72	10.8	3.2
19	1.7041	1.75	11.5	3.4
20	1.7173	1.75	12.1	3.3

Figure 4.3 : Linear Array Results

The successive over-relaxation method can be effective for improving the convergence rate of linear systems of equations but it is shown in the next section to be of limited use for nonlinear systems of equations. A new technique for improving the convergence rate of relaxed systems of equations which has been found to be useful for nonlinear as well as linear systems of equations has been implemented in the SPLICE2 program. The method is heuristic and is based on an attempt to improve the estimate of the true Norton contribution of a branch of a circuit element to the node to which it is connected. During the iterative solution outlined above for the nonlinear Gauss-Seidel method,

$$\text{solve: } g_i\,(x_1^{k+1}, \cdots x_i^{k+1}, \quad \cdots x_N^k) = 0 \text{ for } x_i^{k+1}, \tag{4.20}$$

all of the variables except x_i^{k+1} are held constant during the solution of the linearized equation. In physical terms, this corresponds to assuming during the solution that all of the neighboring nodes that affect the node whose voltage is being computed (have terms in the equation) are constant voltage sources with no resistance. However, in common situations in the network, the neighboring node may actually have a Thevenin equivalent with a very large equivalent resistance. If the sum of all of the conductances contributed by each branch incident on a given node is stored and used as an estimate of the Thevenin/Norton conductance to ground, it can be used to improve the estimate of the conductance contributions to neighboring nodes of branches that connect them. The standard approach corresponds to using a zeroth order model for the contribution of neighboring nets whereas the new method corresponds to using a first order model for the contribution.

For networks of resistors and current sources, if v_i is the voltage at node i, g_{ij} is the conductance of the branch connecting node i and node j, and b_{ij} is the current source between node i and node j, then Kirchoffs Current Law for node i can be written:

$$\sum_{j=0}^{N} (\, g_{ij}(v_i - v_j) + b_{ij} \,) = 0 . \tag{4.21}$$

or

$$v_i \sum_{j=0}^{N} g_{ij} = \sum_{j=0}^{N} (g_{ij} v_j - b_{ij}) \tag{4.22}$$

where node 0 is the reference node (ground) and associated reference directions are used for b_{ij} so that a positive value for b_{ij} corresponds to current leaving node i. The terms on the right-hand side of Equation 4.22 are constant with respect to v_i and therefore can be thought of as current sources while the sum of conductances multiplying v_i on the left-hand side is the equivalent conductance. Equation 4.22 can be rewritten to solve for v_i

$$v_i = \frac{\sum_{j=0}^{N} (g_{ij} v_j - b_{ij})}{\sum_{j=0}^{N} g_{ij}} . \tag{4.23}$$

The new first order methods (hereafter referred to as *coupling* methods) involve storing the equivalent conductance, $\sum_{j=0}^{N} g_{ij}$, at each node and then subsequently using that value to modify the values of g_{ij} and b_{ij} in a consistent way that improves the rate of convergence. The condition for consistency of the algorithm is that at the point of convergence, the solution of the equivalent network with modified g_{ij} and b_{ij} must give the same solution for the node voltages as the original network. If the modified conductances and currents are respectively $\hat{g}_{ij}$ and $\hat{b}_{ij}$, and $\hat{g}_{ij} > 0$ whenever $g_{ij} > 0$, then this condition is satisfied if:

$$\hat{b}_{ij} = b_{ij} + (v_i^{\,k} - v_j)(g_{ij} - \hat{g}_{ij}) . \tag{4.24}$$

In terms of $\hat{b}$ and $\hat{g}$, Equation 4.23 becomes:

$$v_i^{\,k+1} = \frac{\sum_{j=0}^{N} (\hat{g}_{ij} v_j - \hat{b}_{ij})}{\sum_{j=0}^{N} \hat{g}_{ij}} . \tag{4.25}$$

Note that $\hat{b}_{ij}$ in Equation 4.25 is a function of $v_i^{\,k}$. Thus, Equation 4.25 is true only if the sequence $v_i^{\,k}$ converges to some constant value. To show that Equation 4.25 is equivalent to Equation 4.23, first expand $\hat{b}_{ij}$ in Equation 4.25 to give:

$$v_i^{k+1} = \frac{\sum_{j=0}^{N} (\hat{g}_{ij} v_j - b_{ij} - (v_i^{k} - v_j)(g_{ij} - \hat{g}_{ij}))}{\sum_{j=0}^{N} \hat{g}_{ij}} \tag{4.26}$$

and rearranging gives

$$v_i^{k} \sum_{j=0}^{N} g_{ij} + (v_i^{k+1} - v_i^{k}) \sum_{j=0}^{N} \hat{g}_{ij} = \sum_{j=0}^{N} (g_{ij} v_j - b_{ij}) . \tag{4.27}$$

Setting $v_i^{k} = v_i^{k+1} = v_i$, cancelling terms and rearranging yields Equation 4.23.

The consistency condition given above yields the interesting result that as long as the iteration sequence converges, any value of $\hat{g}_{ij} > 0$ is acceptable as a replacement for g_{ij}. This leaves a great deal of freedom in designing algorithms that are coupling methods. The goal is to reduce the total number of iterations required to achieve convergence. Physical intuition suggests that using a better approximation for the nonzero Thevenin resistance to ground at neighboring nodes should improve the direction gradient for iteration. Thus, $\hat{g}_{ij}$ should be reduced by some amount compared to g_{ij}. One heuristic method which will be referred to as Coupling Method A, uses the sum of the branch conductances as an estimate of the equivalent conductance to ground. If $g_i = \sum_{j} \hat{g}_{ij}$, then Coupling Method A prescribes a $\hat{g}_{ij}$ while solving for v_i of

$$\hat{g}_{ij} = \frac{1}{\frac{1}{g_{ij}} + f \frac{1}{g_j}} \tag{4.28}$$

where f is a heuristically chosen *coupling factor* and g_j has been computed at node j during a previous step. Note that the value of g_j used in Equation 4.28 is not the same as g_i and therefore $\hat{g}_{ij} \neq \hat{g}_{ji}$.

A comparison of four acceleration methods is given in Figure 4.4 for the resistor arrays where "GS" stands for the simple Gauss-Seidel iterative method, "sor" stands for the ω in Gauss-Seidel with over-relaxation, "coup" stands for the coupling factor f used in Coupling Method A, and "excoup" stands for the coupling factor f used in Coupling Method B, to be described later. Figure 4.4 indicates that all three acceleration methods are effective for the linear array, but that successive over-relaxation is more effective for higher order problems.

Coupling Method B is a heuristic coupling method which yields an exact value for the conductance to ground, g_i , for a restricted class of networks. A branch with conductance $g_{i\,j}$ between nodes i and j is shown in Figure 4.5 along with the grounded Norton equivalent representation for the nodes. The exact value of g_i is defined to be

Minimum of Maximum DC Iterations							
N	GS	sor	num	coup	num	excoup	num
1,3	1	1	1	0	1	0	1
1,4	7	1.1	5	0.4	5	0.2	6
1,5	12	1.2	7	0.5	7	0.3	8
1,6	18	1.3	10	0.6	10	0.4	9
1,7	25	1.4	12	0.6	11	0.4	11
1,8	30	1.4	15	0.8	12	0.5	14
1,9	39	1.5	16	0.7	15	0.5	17
1,10	47	1.5	19	0.7	20	0.5	17
1,11	54	1.6	22	0.7	22	0.5	18
1,12	62	1.6	20	0.8	20	0.5	22
1,13	67	1.6	24	0.8	24	0.5	25
1,14	75	1.7	24	0.8	28	0.5	26
1,15	83	1.7	27	0.8	27	0.5	29
1,16	90	1.7	27	0.8	32	0.5	33
1,17	98	1.7	31	0.8	35	0.5	36
1,18	104	1.7	33	0.8	37	0.5	39
1,19	113	1.7	36	0.8	39	0.5	43
1,20	119	1.7	39	0.8	42	0.5	46
2,2	11	1.2	6	0.5	6	0.3	7
2,3	41	1.5	13	0.8	21	0.7	11
2,4	82	1.7	21	0.8	47	0.9	26
2,5	130	1.7	34	0.8	80	0.9	43
2,6	184	1.7	35	0.8	117	0.9	71
2,7	237	1.8	49	0.8	158	0.9	102
2,8	288	1.9	63	0.8	202	0.9	133
2,9	335	1.9	53	0.8	243	0.9	163
2,10	382	1.9	55	0.8	285	0.9	198
3,2	25	1.4	9	0.8	14	0.7	10
3,3	99	1.7	22	0.8	69	0.9	52
3,4	210	1.8	38	0.8	155	0.9	127
3,5	341	1.9	55	0.8	263	0.9	226

Figure 4.4 : Comparison of Acceleration Methods

$$g_i = \frac{\partial v_i}{\partial b_i} \tag{4.29}$$

where ∂b_i is defined to be an infinitesimal injection of current from the reference node to node i. It is possible to find a value for $\hat{g}_{ij}$ such that, at convergence, the value $g_i = \sum \hat{g}_{ij}$ is exact if the branch associated with g_{ij} between nodes i and j is a cut-set branch i.e., there are no other paths between the nodes. The formula for finding a value for $\hat{g}_{ij}$ which is the correct Norton equivalent conductance for branch ij is derived as follows. If $g_{id} = g_i - \hat{g}_{ij}$ and $g_{jd} = g_j - \hat{g}_{ji}$ are the values of equivalent conductance to ground with the branch ij deleted, then, $\hat{g}_{ij}$ and $\hat{g}_{ji}$ are computed as follows:

$$\hat{g}_{ij} = \frac{1}{\dfrac{1}{g_{ij}} + \dfrac{1}{g_{jd}}} \tag{4.30a}$$

and

$$\hat{g}_{ji} = \frac{1}{\dfrac{1}{g_{ji}} + \dfrac{1}{g_{id}}} . \tag{4.30b}$$

Substituting for $\hat{g}_{ij}$ and $\hat{g}_{ji}$ and noting that $g_{ij} = g_{ji}$ gives

$$g_{id} = g_i - \frac{1}{\dfrac{1}{g_{ij}} + \dfrac{1}{g_{jd}}} \tag{4.31a}$$

and

$$g_{jd} = g_j - \frac{1}{\dfrac{1}{g_{ij}} + \dfrac{1}{g_{id}}} . \tag{4.31b}$$

Equations 4.31a and 4.31b can be solved to find

$$g_{id} = \frac{g_i}{2} - g_{ij} \pm \frac{1}{2} (g_i^{\,2} + \frac{4 g_i g_{ij}^{\,2}}{g_j})^{\frac{1}{2}} \tag{4.32a}$$

and

$$g_{jd} = \frac{g_j}{2} - g_{ij} \pm \frac{1}{2} (g_j^{\,2} + \frac{4 g_j g_{ij}^{\,2}}{g_i})^{\frac{1}{2}} . \tag{4.32b}$$

The plus sign in Equation 4.32 is used since if g_{ij} is zero, deleting the branch has no effect and $g_{id} = g_i$. Heuristic Coupling Method B is then completed by setting

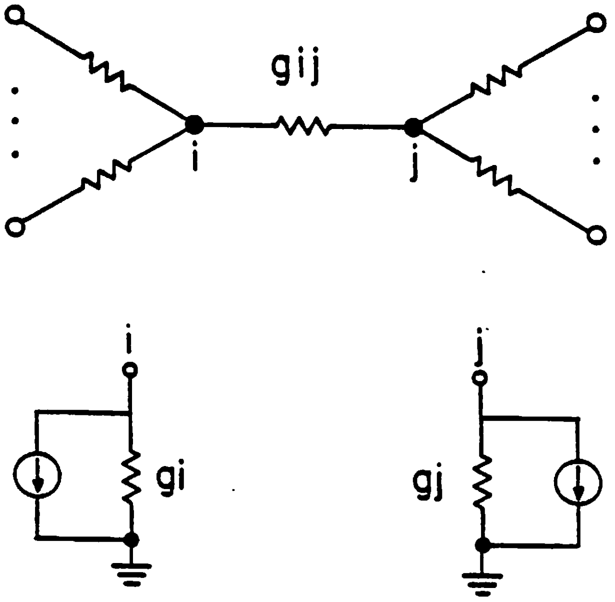


Figure 4.5 : Norton Equivalents at Nodes

$$\hat{g}_{ij} = \frac{1}{\frac{1}{g_{ij}} + f\frac{1}{g_{jd}}} \tag{4.33}$$

where f is the coupling factor and g_{jd} is computed from Equation 4.32. A coupling factor of $f = 1$ corresponds to using "exact" coupling. Note that Equation 4.32 contains a square root which requires significantly more computation time than simple arithmetic operations. Since the method is heuristic, there may be a simple way of improving the estimate in Coupling Method A so that it is comparable to Coupling Method B while still not requiring the computation of a square root.

A sequence of plots is shown in Figure 4.6 which illustrates the convergence to a solution of the relaxation process. The plots are three-dimensional plots of voltage on the vertical Z-axis, iteration number on the horizontal Y-axis, and node number on X-axis which extends out of the paper. This style of plot will be used many times in this section to illustrate the nature of the convergence behavior of various algorithms. It is important to note that the axes have been normalized so that the maximum value of all three dimensions is one. Thus, if the number of iterations in a modified algorithm is reduced by half, this fact must be ascertained from the plot legend, not the plot. The results shown in Figure 4.6 are for a linear array of seven 1Ω resistors connected between a fixed voltage source of 10V and ground. In Figure 4.6a, no acceleration of any kind is used and convergence requires 26 iterations. In Figures 4.6b-d, Coupling Method A is illustrated with a varying coupling factor. The optimum acceleration is achieved with a coupling factor of 0.7 which requires 11 iterations for convergence and is shown in Figure 4.6b. When the coupling factor is increased beyond the optimum value of 0.7, the iteration matrix acquires complex eigenvalues as can be seen by the peaking in Figure 4.6c (coupling factor 0.8) and the ringing in Figure 4.6d (coupling factor 1.0). In the tables of Appendix C, non-convergence is indicated with an iteration number of -1. Non-convergence occurs when at least one eigenvalue of the modified system of equations lies on or outside the unit circle.

The simple example of Figure 4.6 is uniform and presents no difficulties to solution. Using a value of 1Ω for the center resistor and a value of 100Ω for the other six resistors results in a much more difficult example illustrated in Figure 4.7. This difficult example requires 294 iterations to converge without the use of acceleration methods. This slow convergence rate can be understood by considering the third node from the voltage source, which is connected to both a 1Ω and 100Ω resistor. Before the first iteration, all of the node voltage values are zero. As the voltage at the second node rises, it drives current into the node 3 through the 100Ω resistor. However, during the solution of node 3, node 4 is still at 0V and, with a zero coupling factor, is approximated by a voltage source with no resistance (when in fact it has approximately 300Ω of resistance to ground). Thus, the current injected into node 3 is extremely ineffective at raising the voltage on node 3 since it appears to be connected to ground through a 1Ω resistor. The use of a coupling factor greater than zero improves the approximation of the resistance to ground at node 4 so that the current from node 2 is more effective at raising the voltage of node 3.

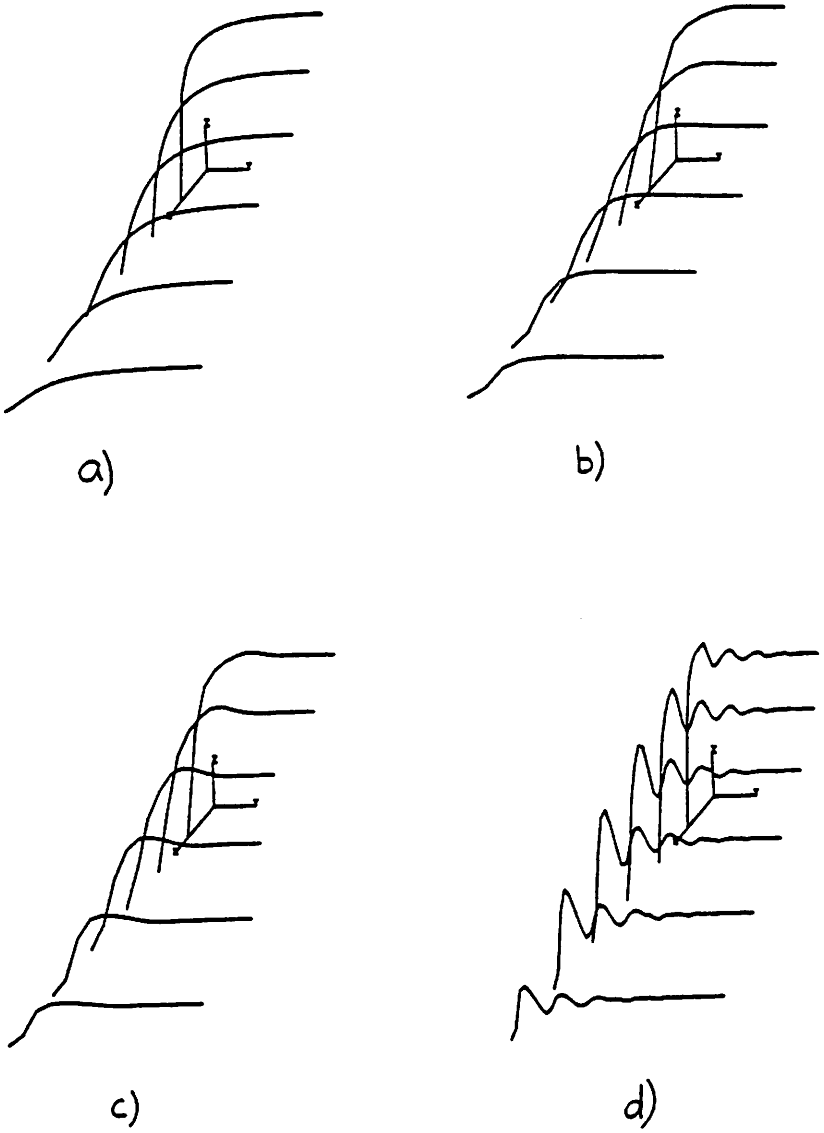


Figure 4.6 : Iterate Plots for Resistors

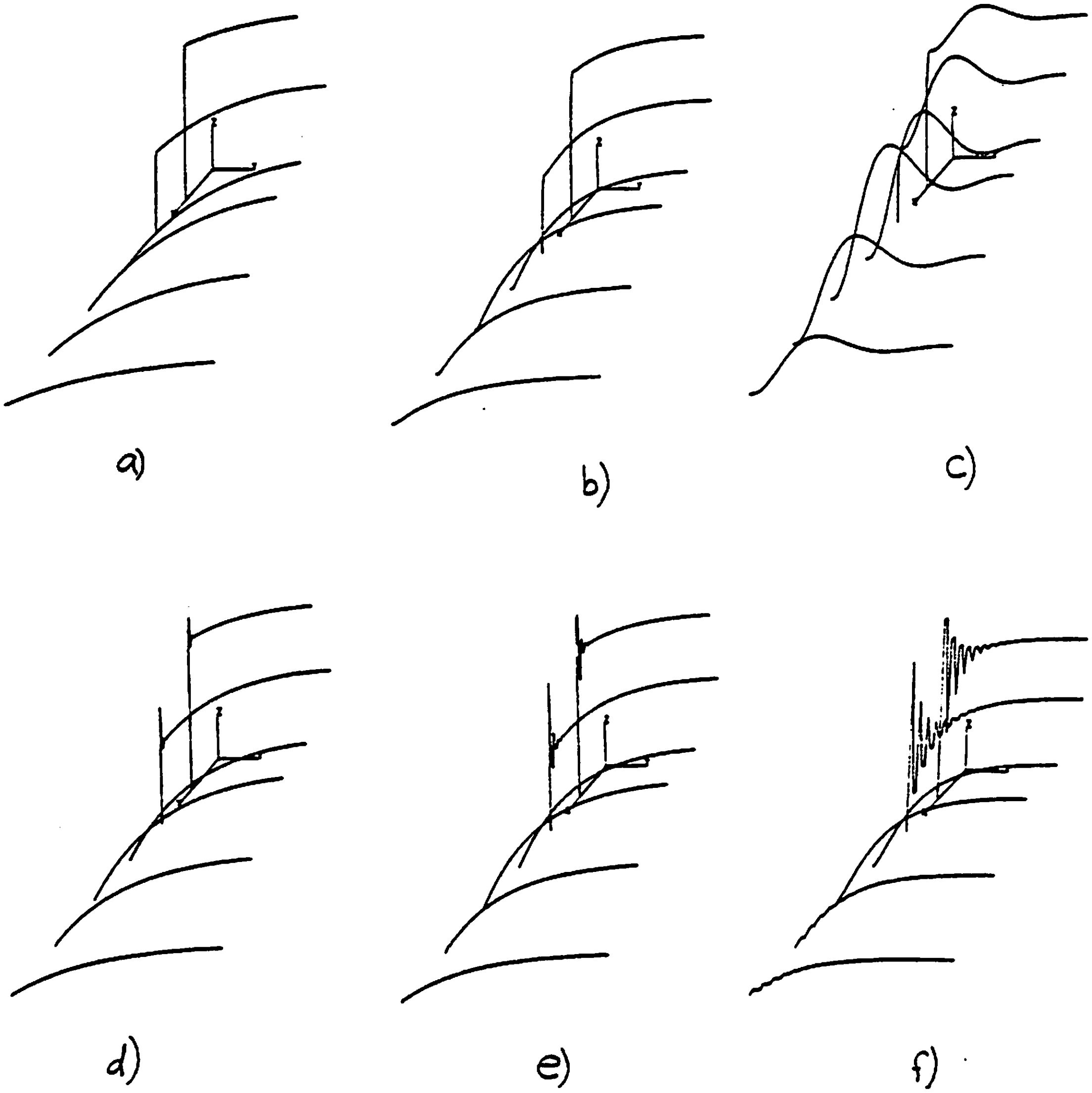


Figure 4.7 : Difficult Resistor Example

The optimum coupling factor for this problem is 0.5 as shown in Figure 4.7c and requires 64 iterations for convergence which is approximately a five-fold improvement over no coupling. The iteration "waveform" can be seen in Figure 4.7c to exhibit the effect of complex eigenvalues. Coupling factors of 0.6 and larger are unstable for this example. Also shown in Figures 4.7d-f are the iteration "waveforms" for this example using the SOR acceleration method. The optimum value of ω for this example is 1.8 which is shown in Figure 4.7f. The fast ringing in the plot of Figure 4.7f suggests that a value of ω that is a function of iteration could yet improve the convergence rate. The Chebyshev semi-iterative method [61] is an example of a *non-stationary* algorithm (one that changes during iteration) where the SOR factor (ω) is variable.

Convergence data for the difficult example of Figure 4.7 are summarized in Figure 4.8 for SOR, Coupling Method A, and Coupling Method B algorithms where "iter" is the number of iterations until convergence is attained and "maxerr" is the maximum of the errors from the true solution after the last iteration. The difficult example also exhibits the false convergence problem that is described in Chapter 3, where the per-iteration change in the voltage

SOR			Coupling Method A			Coupling Method B		
ω	iter	maxerr	f	iter	maxerr	f	iter	maxerr
1.0	294	0.77V	0.0	294	0.77V	0.0	294	0.77V
1.1	262	0.65V	0.1	260	0.64V	0.1	233	0.54V
1.2	232	0.54V	0.2	221	0.50V	0.2	144	0.27V
1.3	204	0.44V	0.3	173	0.34V	0.3	-1	
1.4	176	0.35V	0.4	113	0.18V			
1.5	149	0.27V	0.5	64	0.13V			
1.6	122	0.20V	0.6	-1				
1.7	94	0.14V						
1.8	64	0.07V						

Figure 4.8 : Data for Difficult Example

falls below the tolerance before reaching the asymptotic value within the specified accuracy. This can be seen in Figure 4.7a where the upper iteration curve stops short compared to the other curves. Figure 4.8 shows that as the convergence rate is increased, there is a concomitant increase in the accuracy. None of the errors reported in Figure 4.8 is less than 0.1% as specified. Clearly more work is needed to combine acceleration methods and convergence detection methods.

4.2.2. Nonlinear Equations

The solutions of systems of nonlinear equations using relaxation have different properties from the solution of nonlinear equations using the conventional technique of linearization (Newton-Raphson) followed by direct solution of the resulting linear equations. Even with conventional methods, there is some variation on how the linearization is performed. The process of linearizing Equation 4.4 to Equation 4.5,

$$g(x)=0 \rightarrow A\ x = b\ , \tag{4.34}$$

is usually damped using a limiting algorithm [4]. Limiting algorithms help to prevent numerical over- or under-flow and keep the state of the system of equations from ending up at extremely non-physical values. The Newton-Raphson algorithm corresponds to the use of the best linear model for the nonlinear devices, but other methods, such as the Secant Method, Line-Through-Origin Method, and the Chord Method, use different ways of approximating the nonlinearity [63]. All of these methods must obey the consistency condition from Section 4.2.1 that, when the terminal voltages have converged, the modified model must give the same terminal branch currents as the exact model. It can be shown that Newton-Raphson has the highest asymptotic rate of convergence of all of these methods. Model approximations such as the Secant Method or Line-Through-Origin Method could be used with Iterated Timing Analysis, however the fast convergence rate of the Newton-Raphson method is even more important for relaxation. It is shown below that the new coupling techniques given in the previous section for linear problems can also be used to provide considerable speedup of the convergence rate for nonlinear problems.

A nonlinear example made up of an array of six diodes connected at one end to a 10 V constant voltage source through a 1 Ω resistor and at the other end connected to ground is shown in Figure 4.9. This example is used to compare the effectiveness of different convergence acceleration methods for nonlinear problems. In particular, two limiting algorithms for the diodes (branch current limiting), a node voltage limiting algorithm, and the coupling method from the last section are compared separately and in combination with each other. The coupling method is Coupling Method A described above and is applied in the diode to the linearized conductance as if it were an ordinary resistor. For the diode branch current limit-

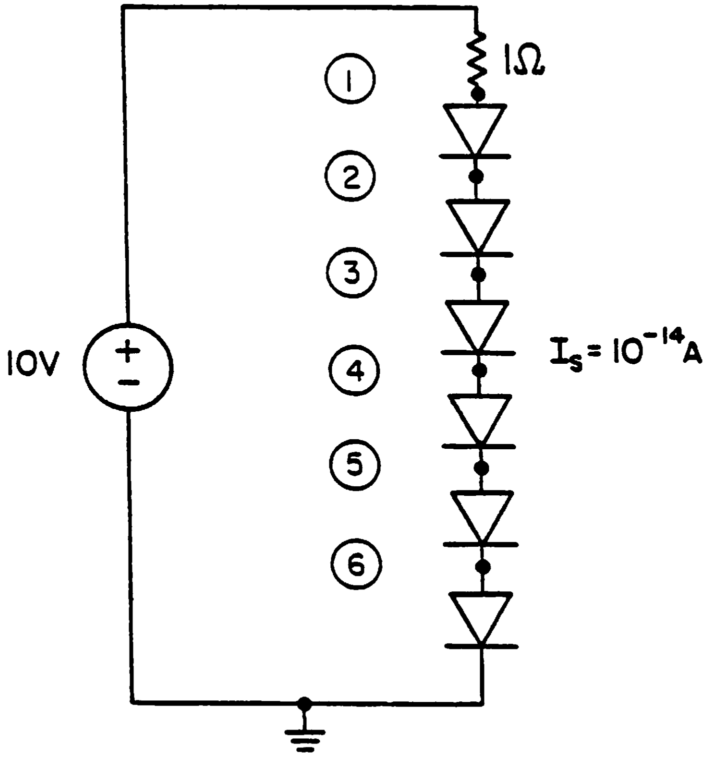


Figure 4.9 : Diode Nonlinear Example

ing, if vnew is the new terminal voltage for the diode, vold is the last terminal voltage that was used after limiting was applied, vt is the thermal voltage (kT / q), and vcrit is defined as the voltage at which the diode characteristic has the most curvature (the knee of the curve), then the diode limiting algorithm referred to as "old" can be written:

```
if ( vnew > vcrit ) {
   vlim = 2 * vt;
   delv = vnew - vold;
   if ( ABS(delv) > vlim ) {
      if ( vold ≤ 0.0 ) {
         vnew = vt * log( vnew/vt );
      }
      else if ( (1.0 + delv/vt) > 0.0 ) {
         vnew = vold + vt * log( 1.0 + delv/vt );
      }
      else {
         vnew = vcrit;
      }
   }
}
```

and the diode limiting algorithm referred to as "new" can be written:

```
if ( vnew > vcrit ) {
   vlim = 2 * vt;
   delv = vnew - vold;
   if ( ABS(delv) > vlim ) {
      if ( vold ≤ 0.0 ) {
         vnew = vt * log( vnew/vt );
      }
      else if ( vold < vcrit ) {
         vnew = vcrit;
      }
      else if ( (1.0 + delv/vt) > 0.0 ) {
         vnew = vold + vt * log( 1.0 + delv/vt );
      }
      else {
         vnew = vcrit;
      }
   }
} .
```

The difference between the two algorithms is small, but the difference in performance in some situations can be substantial, as will be seen later. The node voltage limiting algorithm is based on [64]. If delta_v is the difference between the node voltage computed at the current iteration and the previous iteration, delviterlimit is the value of the change in voltage above which node voltage limiting will be used. and vold is the voltage for the node at the previous iteration, then the node voltage limiting algorithm can be written

```
if ( uselimiting ) {
  if ( ABS(delta_v) > delviterlimit ) {
    vnew = vold + SIGN(delta_v) *
      (delviterlimit + log10( ABS(delta_v) / delviterlimit ));
  }
} .
```

The results of a series of 20 experiments performed using the SPLICE2 program with the diode example are summarized in Figure 4.10 and the "convergence waveform" plots are given in Figure 4.11. The individual experiments are labelled E0 through E20. These labels are used below to refer to the data in Figure 4.10 and the corresponding plots in Figure 4.11. The diode array example is a difficult one due to the exponential nonlinearity of the diodes and the resulting tight coupling between the nodes. At the operating point, when each diode is conducting, the diodes closely resemble floating voltage sources with a voltage drop equal to the "on" voltage of each diode. Experiment E1 has been performed without any damping or convergence acceleration methods. Two distinct regions can be identified in the plot for E1. The first is the slow linear convergence of voltage that was observed for the single diode with resistor in Chapter 3. The second is a slow asymptotic convergence region which is fairly flat. The limiting algorithm for diodes from SPICE2 is used in E2, where it is seen that the number of iterations is reduced substantially and now more of the iterations are spent in the asymptotic region. Simple node limiting is used in E3 but instead of converging slowly from above, Node 1 converges slowly from below, thus, node limiting alone is not effective at getting the solution into the asymptotic region. However, Experiment E4 shows that node limiting in combination with diode branch limiting is effective at bringing the solution quickly to the asymptotic region. The new diode branch limiting algorithm is compared to the old algorithm in E5 and E6 which correspond to E2 and E4, respectively. It is seen that node limiting alone does not help the new algorithm as much as the old algorithm. So far, none of the methods have significantly improved the rate of convergence in the asymptotic region. In Experiments

E7, E8, E9, and E10, the effect of coupling on the convergence rate with and without node limiting is shown. These experiments show that the addition of the coupling method significantly improves the rate of convergence. Further, E9 compared with E11 shows that the new limiting algorithm is significantly more damped than the old one by the absence in E9 of the extra peaks that appear in E11. Experiments E12-E20 show various combinations of the three methods. The best results are for E16 which requires only 30 iterations and makes use of all three acceleration methods. This is close to a factor of 10 fewer iterations than without any acceleration methods at all.

Diode Chain (6) Convergence Rate				
plot	node limiting	diode limiting	couple factor	iter
E1	no	no	0.0	249
E2	no	old	0.0	141
E3	yes 0.5V	no	0.0	240
E4	yes 0.5V	old	0.0	96
E5	no	new	0.0	131
E6	yes 0.5V	new	0.0	166
E7	no	new	0.3	99
E8	yes 0.5V	new	0.3	97
E9	no	new	0.7	58
E10	yes 0.5V	new	0.7	56
E11	no	old	0.7	84
E12	yes 0.5V	old	0.7	56
E13	yes 0.5V	old	0.8	60
E14	yes 0.2V	old	0.7	111
E15	yes 0.5V	new	0.8	54
E16	yes 0.8V	new	0.7	30
E17	yes 1.1V	new	0.7	52
E18	yes 1.1V	new	0.8	47
E19	yes 0.8V	new	0.8	46
E20	yes 0.8V	old	0.7	69

Figure 4.10 : Convergence Data for Diode Example

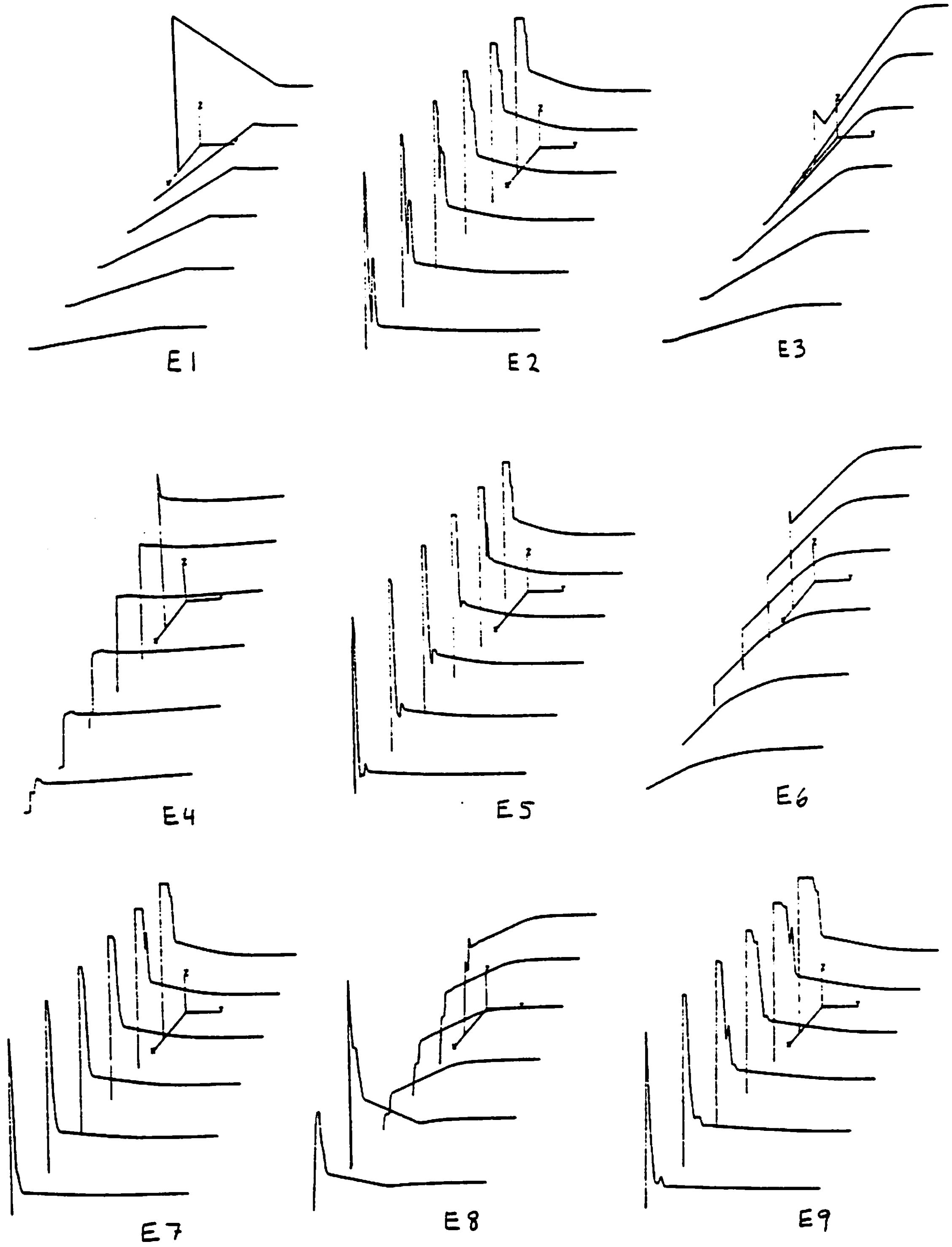
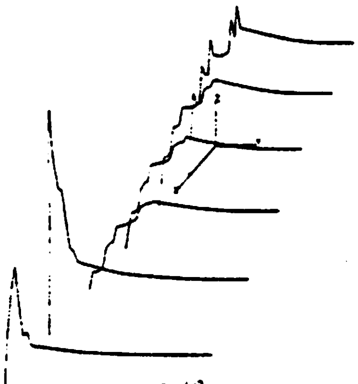
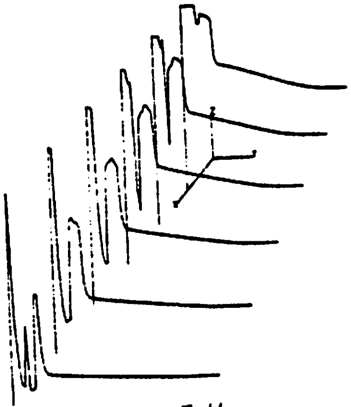


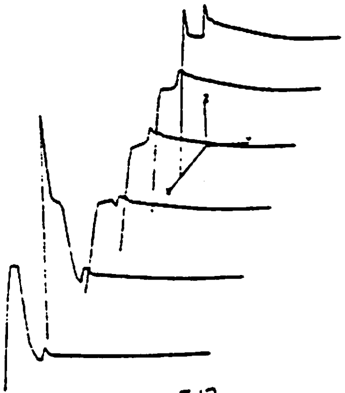
Figure 4.11 : Convergence Plots for Diode Example



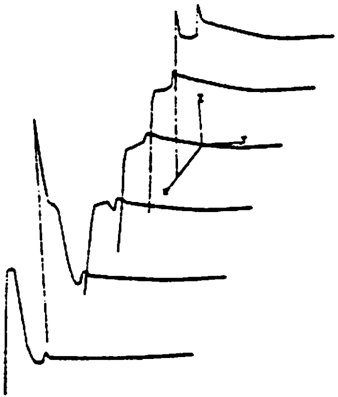
E10



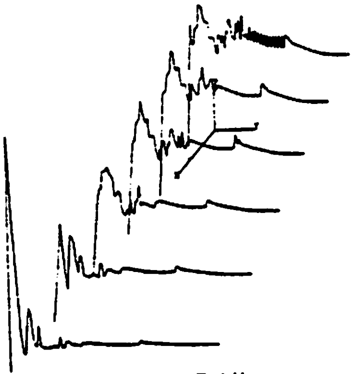
E11



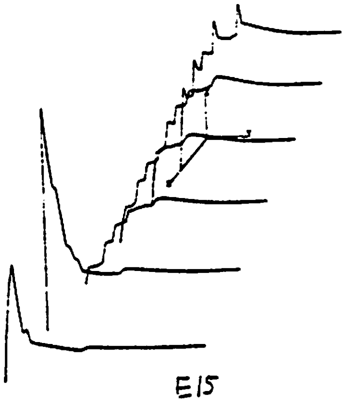
E12



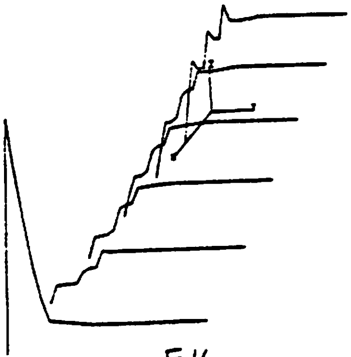
E13



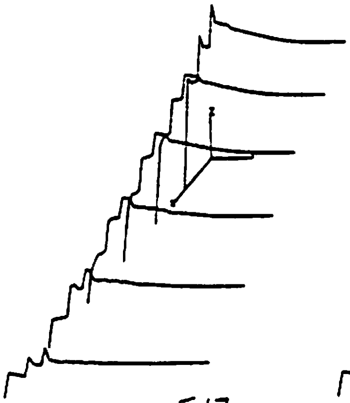
E14



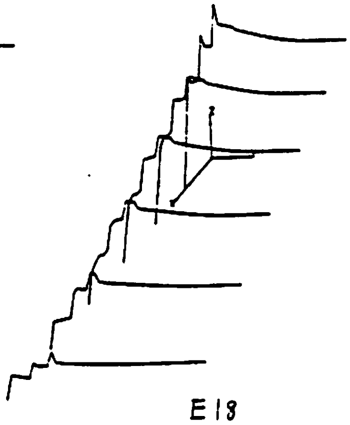
E15



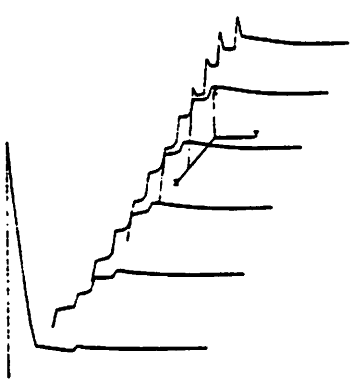
E16



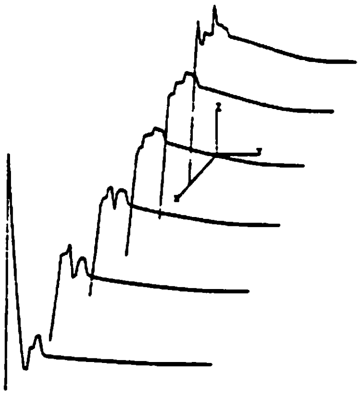
E17



E18



E19



E20

The results from above clearly indicate that convergence acceleration methods can make a very large difference in the number of iterations required for convergence of nonlinear equations. For the diode array example, combinations of methods are more powerful than any single method applied by itself. The results also show that limiting algorithms are useful for helping the solution to reach the asymptotic convergence region more quickly. Coupling methods further accelerate the convergence rate in the asymptotic region. It is unlikely that the speedups observed for an arbitrary network found in practice would be as large as found for this example, since the optimum parameters are likely to be circuit and technology dependent, however the average network would not likely be as difficult as this chaing of diodes. There is likely to be some relationship between the fact that the best node limiting occurs with a value of 0.8V which is close to the diode "on" (or "critical") voltage. This dependence is accentuated by the way the diodes are connected in series, and an arbitrary network of MOS transistors would probably behave differently. Nevertheless, further work on how these methods interact and on new acceleration methods is likely to be fruitful. Further, techniques such as node limiting, improved diode branch limiting, and coupling methods could be applied to standard circuit simulators such as the SPICE2 program.

At this point, it is worth noting again the similarity between the "iteration waveform" and a time-domain transient waveform where the iteration number is considered to be discrete time. The primary goal of convergence acceleration methods is to speed the response of the iteration waveform without regard for the actual path that the waveform traces out. Occasionally, the iteration sequence does not converge. One way of improving the stability of the dc convergence process is to include the capacitor elements for the transient simulation and perform a pseudo-transient simulation with the independent voltage sources held constant to find the operating point. This is similar to the method used in the ASTAP program [65]. The truncation error of this pseudo-transient simulation need not be controlled since the details of the iteration waveform are of no concern, so long as it converges. Algorithms that modify the iteration sequence can be represented by pseudo-elements in the iteration domain. Voltage limiting algorithms correspond to pseudo-elements which are capacitive in nature since they tend to oppose changes in voltage. The successive over-relaxation (SOR) method also is capacitive in nature where an SOR factor greater than one corresponds to a negative capacitance and an SOR factor less than one corresponds to a positive capacitance. An SOR factor less than one is similar to "lag compensation" and can stabilize the iteration sequence, as will be shown in the next section.

Figure 4.12 shows three iteration waveforms for the NMOS depletion-load operational amplifier, that is described more fully in Chapter 6, with three different values of SOR factor, ω. The opamp is connected in a unity-gain feedback configuration. Figure 4.12a corresponds to $\omega = 1$, or no under- or over-relaxation and is seen not to converge in the dc iteration. Figure 4.12b corresponds to $\omega = 0.5$ which has an effect similar to lag compensation and the waveform shows a longer oscillation period compared to Figure 4.12b. With a value for ω of

0.33, the iteration waveform is stabilized and requires 59 iterations for solution as is seen in Figure 4.12c.

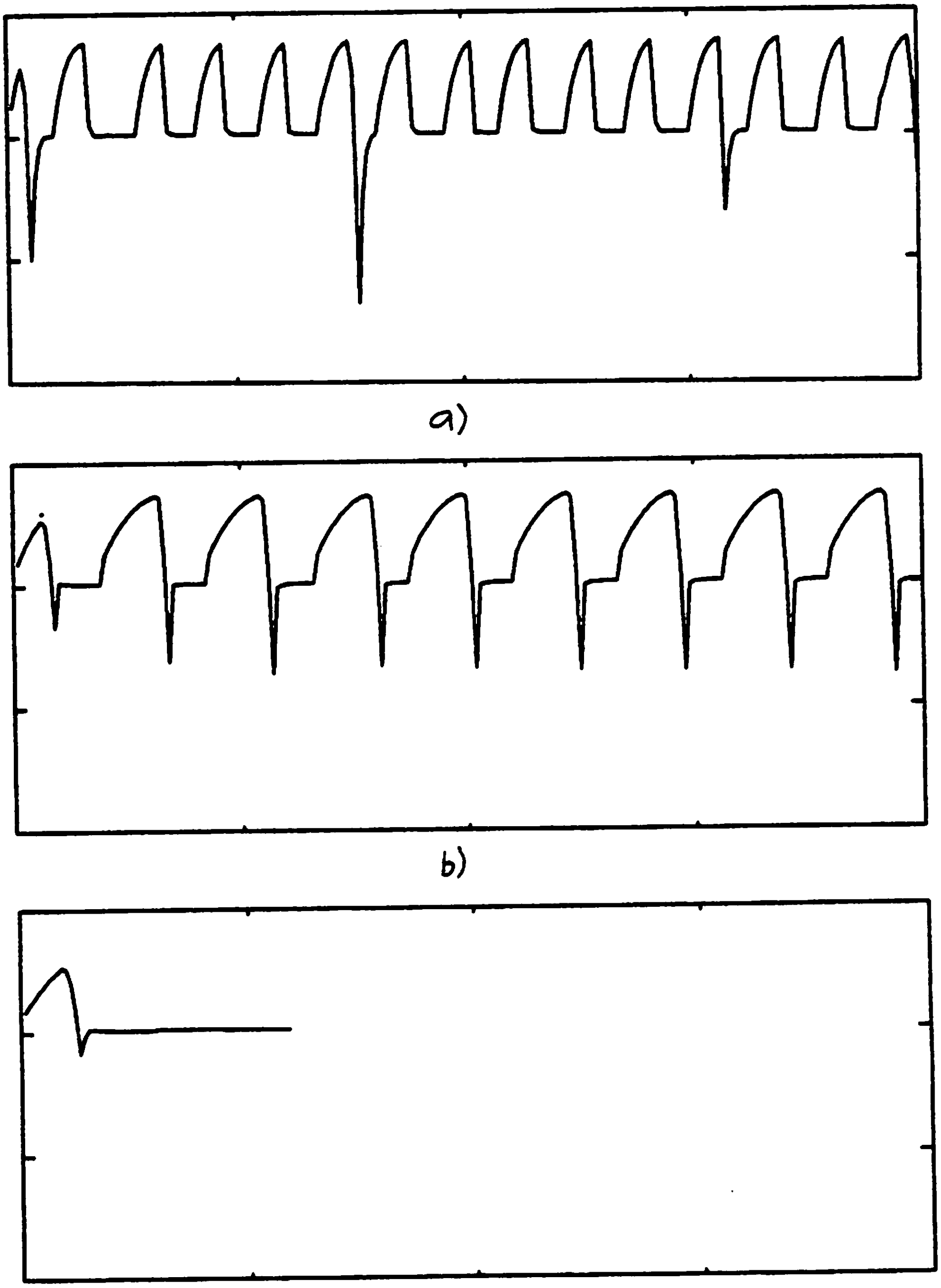


Figure 4.12 : Convergence Plots for Opamp Example

4.2.3. Under-Relaxation

The results of Figure 4.12 suggest that under-relaxation is an effective tool for enlarging the domain of convergence of iterative methods. It is shown in this section under what conditions under-relaxation will stabilize a sequence of iterations along with some ways of estimating the under-relaxation factor. The starting point is with the Gauss-Jacobi iteration method for the solution of linear equations since an explicit upper bound for ω can be calculated easily. Applying under-relaxation to the iteration matrix of Equation 4.7b yields

$$M_{GJ\,\omega} \equiv -\omega D^{-1}((L+U)+(1-\omega)I\ ). \tag{4.35}$$

If $M_{GJ\,1} \equiv -D^{-1}(L+U)$ then Equation 4.35 becomes

$$M_{GJ\,\omega} = \omega M_{GJ\,1} + (1-\omega)I\ . \tag{4.36}$$

If the eigenvalues of the unmodified iteration matrix, $M_{GJ\,1}$, are λ and the eigenvalues of the modified iteration matrix, $M_{GJ\,\omega}$, are λ_ω, then the effect of under-relaxation on the eigenvalues can be computed as follows: Let x be an eigenvector of $M_{GJ\,\omega}$ with corresponding eigenvalue λ_ω so that

$$M_{GJ\,\omega}x = \lambda_\omega x\ . \tag{4.37}$$

Using Equation 4.36 this becomes

$$(\omega M_{GJ\,1} + (1-\omega)I\,)x = \lambda_\omega x \tag{4.38}$$

and rearranging

$$M_{GJ\,1}x = \frac{\lambda_\omega - (1-\omega)}{\omega}x \tag{4.39}$$

$$= \lambda x\ . \tag{4.40}$$

Thus, the modified eigenvalues are related to the original eigenvalues by

$$\lambda_\omega = 1 + \omega(\lambda - 1)\,. \tag{4.41}$$

Using Equation 4.41 it is now possible to prove that under-relaxation enlarges the domain of convergence.

Theorem 4.1: If $M_{GJ\,\omega}$ and $M_{GJ\,1}$ are defined as above and if magnitudes of the original eigenvalues, λ, are all bounded and if the real parts of the original eigenvalues are all strictly less than one, then there exists an $\omega > 0$ such that the modified eigenvalues, λ_ω, are all inside the unit circle i.e., the modified iteration matrix is contractive.

Proof: The proof proceeds constructively to find a value for ω for which the magnitudes of the eigenvalues are less than one. From Equation 4.41 the magnitudes of the eigenvalues λ_ω are

$$|\lambda_\omega| = (\lambda_\omega \overline{\lambda_\omega})^{\frac{1}{2}} \tag{4.42}$$

$$= [\,(1 + \omega(\lambda - 1))\,(1 + \omega(\bar{\lambda} - 1))\,]^{\frac{1}{2}} \tag{4.43}$$

$$= [\,1 + 2\omega(\mathrm{Re}(\lambda) - 1) + \omega^2(\lambda\bar{\lambda} - 2\mathrm{Re}(\lambda) - 1)\,]^{\frac{1}{2}}\,. \tag{4.44}$$

Now let $C \equiv 1 - \mathrm{Re}(\lambda)$ and write $\lambda\bar{\lambda}$ as $|\lambda|^2$ so that

$$|\lambda_\omega| < 1 \tag{4.45}$$

becomes

$$[\,1 - 2\omega C + \omega^2(|\lambda|^2 + 2C - 1)\,]^{\frac{1}{2}} < 1 \tag{4.46}$$

and squaring both sides

$$0 < (\,1 - 2\omega C + \omega^2(|\lambda|^2 + 2C - 1)\,) < 1\,. \tag{4.47}$$

The condition that the real part of the unmodified eigenvalues is strictly less than one implies that $C > 0$. In the case $|\lambda| < 1$, the original matrix is contractive and therefore a value of $\omega = 1$ yields the desired result of $|\lambda_\omega| < 1$. In the case $|\lambda| \geqslant 1$, the term $(|\lambda|^2 + 2C - 1) \geqslant 0$ and so the condition of Equation 4.47 becomes

$$\omega^2(|\lambda|^2 + 2C - 1) < 2\omega C \tag{4.48}$$

and since $\omega \neq 0$

$$\omega < \frac{2C}{|\lambda|^2 + 2C - 1}\,. \tag{4.49}$$

Let $L \equiv \max_{i=1,N} |\lambda_i| < \infty$ be the magnitude of the largest eigenvalue of the unmodified matrix.

Set

$$\omega_{\min} = \frac{2C}{L^2 + 2C - 1} \tag{4.50}$$

and choose the value of under-relaxation factor

$$0 < \omega < \omega_{\min} \tag{4.51}$$

This choice of ω is sufficient to guarantee that all of the eigenvalues lie inside the unit circle, thus completing the proof.

Note that if the value of the largest eigenvalue is real, then Equation 4.51 reduces to

$$0 < \omega < \frac{2}{1 + |\lambda_{\max}|} . \tag{4.52}$$

The effect of under-relaxation on eigenvalues is illustrated in Figure 4.13 where it is seen that as the under-relaxation factor approaches zero, all of the eigenvalues converge on a value of one. If the real part of the eigenvalue is less than one, then it approaches a value of one from the left and thus passes inside the unit circle. If the real part of the eigenvalue is greater than one, then it approaches a value of one from the right and thus remains unstable. The calculation of the point of intersection of the parametric curve for the eigenvalue with the unit circle yields the value of under-relaxation factor which is the strict upper bound for convergence. The constructive proof of Theorem 4.1 calculated the bound for the case of Gauss-Jacobi iteration. The corresponding bound for Gauss-Seidel iteration has not been computed, although the Stein-Rosenberg theorem suggests that a Gauss-Seidel iteration should require a smaller upper bound than for the Gauss-Jacobi iteration since the spectral radius is larger for Gauss-Seidel than Gauss-Jacobi when the spectral radius for the Gauss-Jacobi iteration is larger than one. Thus, Gauss-Seidel diverges more quickly when Gauss-Jacobi diverges and converges more quickly when Gauss-Jacobi converges. This yields the important observation that Gauss-Jacobi iteration should be used when using under-relaxation and Gauss-Seidel should be used when using over-relaxation.

The extension of these techniques to nonlinear systems of equations requires the following definition of the Gauss-Jacobi and Gauss-Seidel iteration matrices [57]. Let $g'(x)$ denote the Jacobian of g computed at x. Let g be continuously differentiable in an open

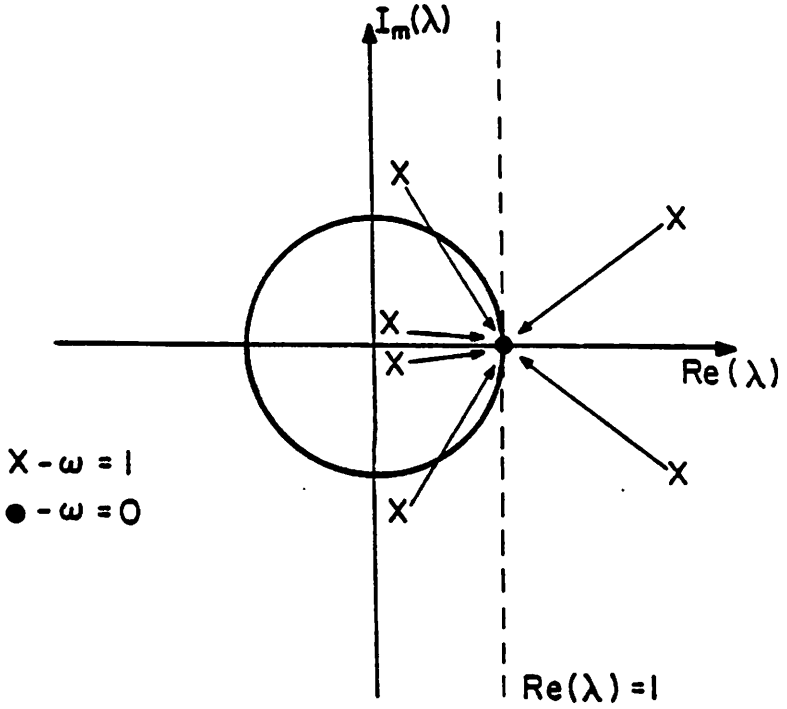


Figure 4.13 : Eigenvalues and Under-Relaxation

neighborhood S_0 of $\hat{x}^*$ for which $g(\hat{x}^*)=0$. Let $g'(\hat{x}^*)$ be split as $L(\hat{x}^*)+D(\hat{x}^*)+U(\hat{x}^*)$ where $L(\hat{x}^*)$, $D(\hat{x}^*)$ and $U(\hat{x}^*)$ are respectively the strictly lower triangular part, the diagonal part and the strictly upper triangular part of $g'(\hat{x})$. Let $M_{GJ}(\hat{x})$ and $M_{GS}(\hat{x})$ be defined as follows:

$$M_{GJ}(\hat{x}^*) = -D(\hat{x}^*)^{-1}(L(\hat{x}^*)+U(\hat{x}^*)) \tag{4.53}$$

and

$$M_{GS}(\hat{x}^*) = -(D(\hat{x}^*)+L(\hat{x}^*))^{-1}U(\hat{x}^*) \tag{4.54}$$

Assume that $D(\hat{x}^*)$ is nonsingular and that all the eigenvalues of $M_{GJ}(\hat{x}^*)$ and $M_{GS}(\hat{x}^*)$ are inside the unit circle. Then there exists an open ball $S \subset S_0$ such that the nonlinear Gauss-Jacobi and the Gauss-Seidel iterations are well-defined and for any $x_0 \in S$, the sequence generated by the iterations converges to $\hat{x}^*$.

Under-relaxation is shown in Theorem 4.1 and Figure 4.13 to be useful for enlarging the domain of convergence for linear systems of equations. Under-relaxation is also useful for enlarging the domain of convergence of nonlinear systems of equations, although proofs of global convergence for nonlinear systems of equations are more difficult. It is possible to prove a result for the case of a one-dimensional nonlinear equation with conditions and bounds similar to those in Theorem 4.1.

Lemma 4.2: Let $g : (\mathbb{R} \rightarrow \mathbb{R})$ be a function for which the following inequality holds in $\mathbb{R}$:

$$g(x) - g(x') \leqslant L(x - x') \quad \text{where} \quad -\infty < L < 1 \tag{4.55}$$

Define $\hat{g}(x)$ as follows:

$$\hat{g}(x) = \omega g(x) + (1-\omega)x \tag{4.56}$$

Then there exists an $\omega : 0 < \omega < 1$ such that

$$|\hat{g}(x) - \hat{g}(x')| \leqslant \gamma |x - x'| \quad \text{where} \quad \gamma < 1 . \tag{4.57}$$

Proof: The proof proceeds by construction. First note that Equation 4.57 states that $\hat{g}(x)$ is contractive. If $-1 < L < 1$, then Equation 4.55 becomes

$$\begin{aligned} |g(x) - g(x')| &\leqslant |L(x - x')| \\ &\leqslant |L| \; |(x - x')| \end{aligned} \tag{4.58}$$

which indicates that $g(x)$ is contractive and thus a value for ω of one is suitable to make $\hat{g}(x)$ contractive. For the case that $-\infty < L \leqslant -1$, the left hand side of Equation 4.57 can be rewritten

$$|\hat{g}(x) - \hat{g}(x')| = |\omega(g(x) - g(x')) + (1-\omega)(x - x')| . \tag{4.59}$$

Further, since $g(x) - g(x') \leqslant L(x - x')$, let

$$g(x) - g(x') = \alpha(x - x') \quad \text{where} \quad -\infty < \alpha \leqslant L \leqslant -1 \tag{4.60}$$

so that Equation 4.57 becomes

$$\begin{aligned} |\hat{g}(x) - \hat{g}(x')| &= |(1 - \omega + \alpha\omega)(x - x')| \\ &= |1 - \omega(1-\alpha)| \; |x - x'| \end{aligned} \tag{4.61}$$

since all of the quantities are real. If ω is chosen small enough:

$$\omega < \frac{2}{1-\alpha} \tag{4.62}$$

then

$$|1-\omega(1-\alpha)| < 1\,. \tag{4.63}$$

Noting that

$$|1-\omega(1-L)| \leqslant |1-\omega(1-\alpha)| \tag{4.64}$$

indicates that if

$$\omega < \frac{2}{1-L} \tag{4.65}$$

then

$$\gamma = |1-\omega(1-L)| < 1 \tag{4.66}$$

and therefore $\hat{g}(x)$ is contractive which completes the proof.

Note that the continuity condition of Equation 4.55 is more strict than the condition for Lipschitz continuity and that $g(x)$ is also Lipschitz continuous with Lipschitz constant $|L|$. The similarity between the two suggests the term *left Lipschitz continuous* to denote that the condition of Equation 4.55 is satisfied. Another way of stating the result of Lemma 4.2 is that if a function, $g(x)$, is everywhere left Lipschitz continuous, then for some $\omega > 0$, $\hat{g}(x)$ is globally contractive using a fixed point iteration. The constraint on L can be weakened to allow $L \geqslant 1$ if there exists an attractive solution and $g(x)$ is left Lipschitz continuous in a closed ball about the point of contraction and containing the starting guess. In this case, the choice of $\omega < \frac{2}{1+|L|}$ will still guarantee that the attractive solution will be found where L is the left Lipschitz constant for the closed ball.

The physical interpretation of this result is that under-relaxation has the effect of damping out the growing oscillations of a solution that is *overly-attracted* to the fixed-point. This procedure of using under-relaxation on a system of equations amounts to a rotation of the intercept of the function hyper-plane with the hyper-plane $g(x) = 1$ until it is guaranteed to intercept everywhere with a slope with magnitude less than one. Figure 4.14 illustrates fixed-point iteration for a linear function, $g(x)$, that is overly-attractive. The fixed-point iteration procedure uses the equation to calculate the variable in terms of itself and then uses that value as the next guess. This is represented in Figure 4.14 as a projection onto a line through the origin with unit slope. Figure 4.14a shows that the overly-attractive function results in an oscillation of increasing amplitude and Figure 4.14b illustrates the use of under-relaxation with a value for ω of approximately 0.4. It is clear from the diagram of Figure 4.14b that under-relaxation corresponds to a rotation of the of the fixed-point function so that it intercepts the line through the origin with a smaller angle. In higher dimensions, the line with unit slope becomes a hyper-plane.

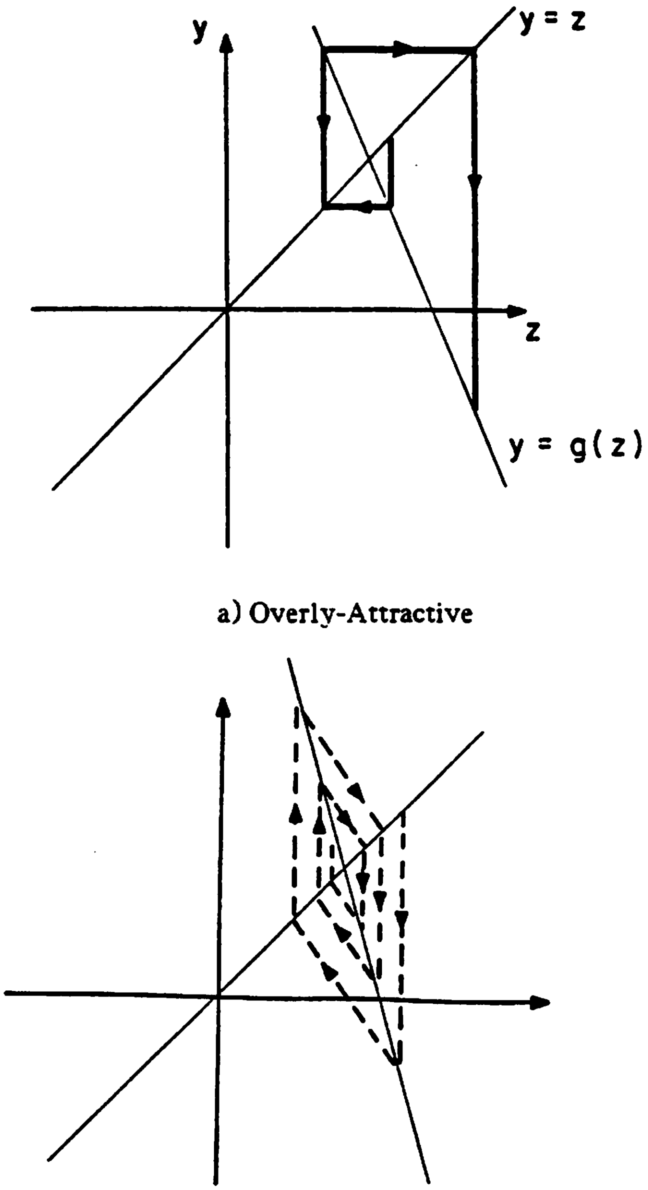
The resistor/diode fixed-point iteration example from Section 3.2.1 that used the "zeroth-order" current source model for the diode shown in Figure 3.5 can be used to illustrate the use of under-relaxation for stabilization and also to show that the condition of Lemma 4.2 can be relaxed somewhat. Repeating Equation 3.9 for the fixed-point iteration here

$$V_2^{k+1} = V_1 - R\, I_s\, (e^{V_2^k / V_t} - 1) \tag{4.67}$$

and taking the derivative of the right hand side of Equation 4.67 gives

$$-\frac{R\, I_s}{V_t} e^{V_2^k / V_t} \, . \tag{4.68}$$

Figure 4.15 shows a plot of Equation 4.67 and its intersection with a line of unit slope using values of $I_s = 10^{-14} A$, $V_t = 26mV$, $R = 1\Omega$ and $V_1 = 2V$. The intersection occurs at the point where $V_2 = 0.841956$ and the slope is -44.54. The magnitude of the single eigenvalue for this equation at the fixed-point is therefore $44.54 \gg 1$ and it can be easily verified from Figure 4.15 that the fixed-point iteration does not converge. In fact, even choosing the starting guess very close to the solution cannot make the iteration stable. Equation 4.68 shows that the magnitude of the derivative grows without bound and so the condition of Lemma 4.2 is not met. However, if the starting guess is 0V, then the derivative is bounded over an interval. Equation 4.65 implies that the upper bound on the under-relaxation factor (ω) such that



a) Overly-Attractive

b) Damped Iteration (ω = 0.4)

Figure 4.14 : Stability of Fixed-Point Iteration

there is some contractive interval about the solution is $\frac{2}{44.54 + 1} = 0.0439$. Figure 4.16 is a plot of the number of iterations required to give a value within 0.1% of the correct answer as a function of ω where non-convergence is indicated with an iteration count of -1 and a starting guess of 0V is used. It can be seen from Figure 4.16 that the fixed-point iteration is stable for $\omega = 0.043$ and unstable for $\omega = 0.044$ as predicted. Let the *critically damped point* be defined to be when the real parts of all eigenvalues are zero. All of the eigenvalues will have a non-negative real part when $\omega \leqslant \frac{\omega_{\max}}{2}$, which is 0.022 for this problem. At the critically damped point, there will be no "overshoot" in the sequence of iterates and plots of the sequence of iterates demonstrate that there is ringing for $\omega = 0.043$ that decreases to become overshoot as ω is decreased to 0.022 whereafter the overshoot disappears. Figure 4.16 also shows that values of ω which are smaller than necessary increase the number of iterations hyperbolically.

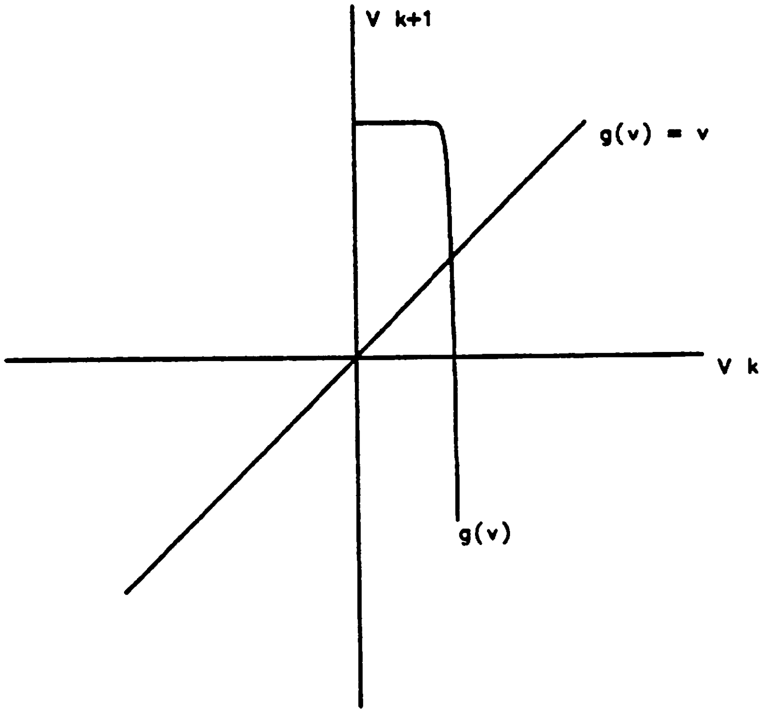


Figure 4.15 : Fixed-Point Iteration of R/Diode

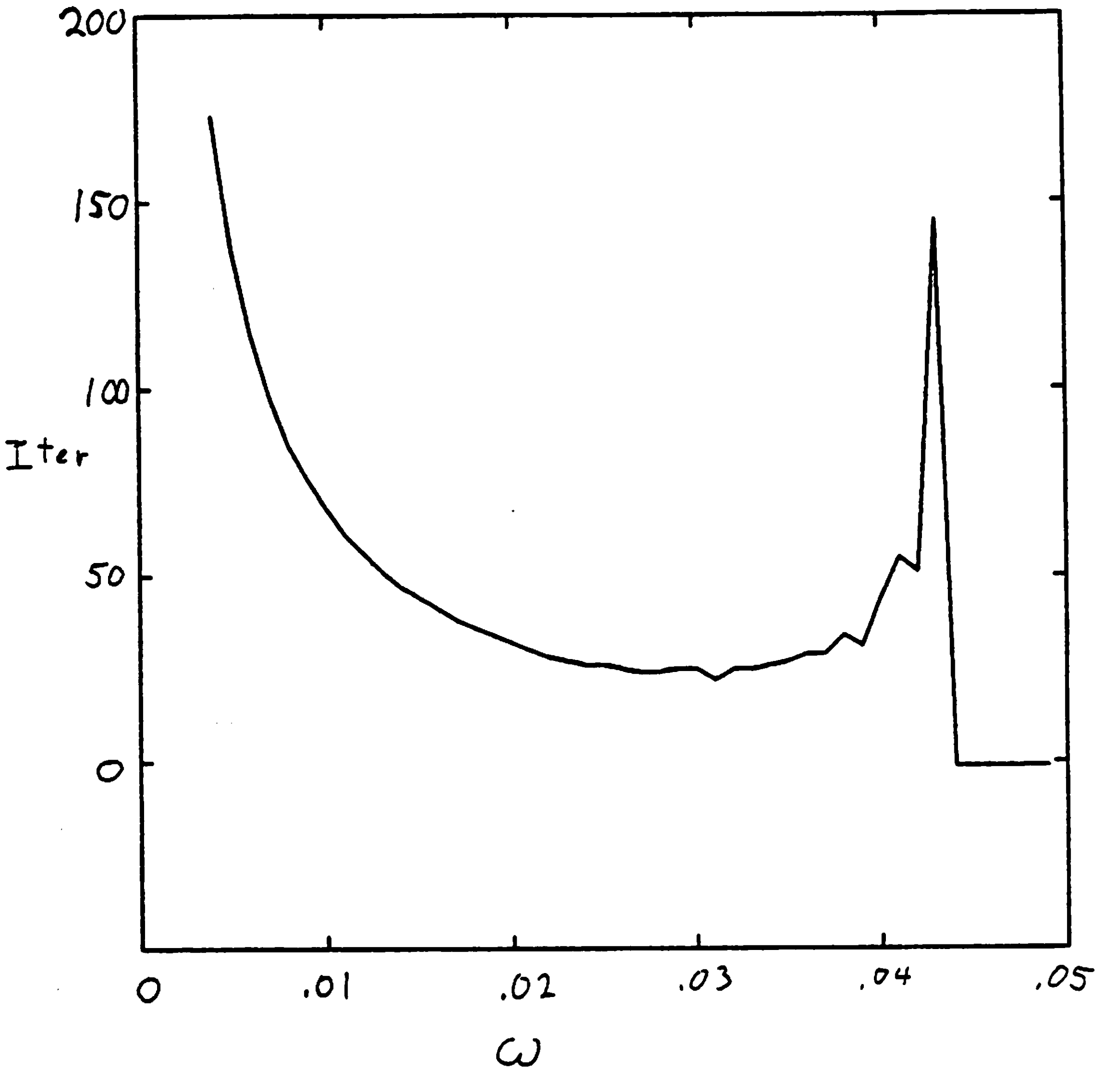


Figure 4.16 : Iterations vs. SOR Factor

4.3. Transient Analysis

The final topic that completes the method of solution of the systems of ordinary partial differential equations is how to control the integration time-step. The method of taking differential equations and obtaining equivalent difference equations by approximating the time derivatives has been presented in Chapter 3. This section gives the details of time-step control as implemented in the SPLICE2 program along with those aspects of integration methods that are of special interest for iterated timing analysis.

In Section 4.2.1, it is mentioned that the condition for stability of convergence of a matrix is that all of the eigenvalues of the matrix have a magnitude less than one (lie inside the unit circle in the complex plane). From the properties of matrix norms, it can be shown that a sufficient condition for all of the eigenvalues of a matrix to have a magnitude less than one is that the matrix be *strictly diagonally dominant*, where strictly diagonally dominant is defined for a matrix, A,

$$|a_{ii}| < \sum_{j \neq i} |a_{ij}| \quad \text{forall } i \in [1,N]. \tag{4.69}$$

It can be proven that under the conditions stated in Section 4.1 that the Gauss-Seidel and Gauss-Jacobi iteration matrices become stable for some time-step greater than zero [53]. The reason is that the capacitance matrix derived from Equation 4.3 is strictly diagonally dominant since the contribution of floating capacitors to the matrix adds equally to the diagonal element and an element off the diagonal but the grounded capacitor at every node adds only to the diagonal elements. As the time-step is made smaller, the contribution of the capacitance matrix to the iteration matrix makes the iteration matrix strictly diagonally dominant and therefore stable. Another way of looking at this fact is that as the time-step is reduced, capacitors become more like voltage sources since they tend to resist a change in voltage across their terminals. Thus, a grounded capacitor at every node will tend to decouple the equations as the time-step is reduced and neighboring nodes begin to appear more like voltage sources.

Because of the grounded capacitors, each transient time-point iteration matrix is more likely to be stable than the iteration matrix for the corresponding purely dc problem. Further, since a good starting guess is easily extrapolated from past values, the trajectory of the solution vector during iteration is likely to be in the asymptotic region of convergence for most of the iterations. For these reasons, obtaining a solution for the transient problem is simpler than obtaining a solution for the dc operating point. The primary problem in the transient domain is how to integrate the elements with memory and choose time-steps such that accuracy and stability are maintained without performing computation that could be avoided.

4.4. ITA in SPLICE2

As pointed out in Section 3.5, iterated timing analysis achieves large speedups due to the presence of *latency* in circuit activity. However, time-step control must be done carefully in order to maintain speed while providing accuracy. An excellent description of the implementation of iterated timing analysis in the SPLICE1 program is given in [3] and is not repeated here. The primary differences between the implementation of the iterated timing analysis algorithm in the SPLICE1 and SPLICE2 programs are:

(1) In the SPLICE1 program, the ***minimum resolvable time*** (MRT) is the numerically smallest interval by which the time variable can change and is specified by the user. The SPLICE2 program allows the value of the minimum resolvable time to be very small in order to achieve accuracy and uses a variable value *local* minimum resolvable time for efficiency.

(2) Time-steps are computed using truncation error estimates and solutions may be rejected. A solution for a time-point that has previously been accepted can be rejected on the basis of truncation error estimates.

(3) The number of previous solutions buffered at each node is specified as a parameter at execution time in the SPLICE2 program. This allows the user to select the window size for the time-step algorithm.

(4) The SPLICE2 program uses a double precision floating point number for the representation of simulation time whereas the SPLICE1 program uses an integer.

(5) In the SPLICE2 program, the nodes are scheduled to be processed whereas in the SPLICE1 program, the lists of nodes to which each element fans out are scheduled.

(6) In the SPLICE2 program, nodes have models in the same fashion that elements have models.

The ITA algorithm is carried out through the execution of the model associated with the electrical-level net. The electrical net model routine is found in the file "elcnet.c" in Appendix A. An iteration of the equation associated with a given net is carried out by one execution of that net's model routine. The rest of this section gives the details of the electrical net model routine. The basic flow of the electrical net routine is:

```
Step 1 Initialize.
Step 2 Compute Norton equivalent of all fanin branches.
Step 3 Compute new voltage.
Step 4 Check iteration tolerance.
Step 5 If not converged, schedule at current time and return.
Step 6 Compute truncation error time-step.
Step 7 If time-step too large, reject step,
           backup time, schedule and return.
Step 8 Schedule at computed time-step in the future and return.
```

The initialization of Step 1 consists of checking to see if this is the first iteration at the current time point. If so, then the number of iterations is set to zero and a predictor is used to compute the starting guess for this time and is pushed onto the solution buffer. If the solution buffer is full, the oldest solution is flushed to disk if the plot flag is set for this net. Step 2 consists of looping over all the fanin branches and setting a signal structure to all zeroes, calling the element model routine associated with each branch, and accumulating the sum of all the currents, absolute value of the sum of all the currents, conductances, and capacitances for all fanin branches. The new voltage is calculated in Step 3 by applying an integration method to the capacitance and solving for the voltage

$$V_{new} = \frac{I_N}{G_N} \, . \tag{4.70}$$

In Steps 4 and 5, the iteration is considered not converged if the per-iteration change in voltage is greater than the voltage tolerance, if the per-iteration change in absolute value of the sum of all the currents is greater than the current tolerance, or if the minimum number of dc or transient iterations have not been taken. If the iteration has not converged, then the values on the net are updated, the net is scheduled to be processed again at the current time, and control returns to the scheduler. Otherwise, the truncation error time-step is computed in Step 6 using either the backward differentiation formula or the divided differences method. In Step 7, the computed time-step is compared with the time-step actually taken and if it is too large or if the number of iterations taken at the current time point is too large then solution at this time is rejected, simulation time is backed up to a time which results in a smaller time-step, the net is scheduled at that time and control returns to the scheduler. Otherwise, in Step 8, the solution is accepted, the time-step window control variables are updated, and the net is scheduled at the current time plus the truncation error time-step in the future.

The time-step window is managed by two control elements called NetElcPreNewStep and NetElcNewStep. Associated with each time window is a minimum time-step stored in ElcMinTimeStep and a maximum time-step stored in ElcMaxTimeStep. At the beginning of simulation the user supplies a value for the number of buffers to be used per window and that value is stored in ElcBufSize. The number of time-step intervals per window is stored in ElcNumIntervals and is smaller than ElcBufSize since integration methods require saving previous state to compute a new state and the solutions back to the beginning of the window might be rejected.

The window mechanism is initialized by setting the ElcMinTimeStep variable to a starting guess provided by the user through the "starttimestep" parameter on the option element and, for the first window, using a window size of 1. The NetElcPreNewStep element is scheduled to be processed before all other physical elements and the NetElcNewStep element is scheduled to be processed after all other physical elements by use of a sorting key for scheduling. The window control elements both are scheduled to be processed at the end of the current time window. The NetElcPreNewStep element calls ElcAdjustNextTimeStep to adjust the time-steps for the next window. The ElcNextMinTimeStep variable is set in the electrical net model to the minimum of itself and the truncation error estimate for that net's time-step. The ElcNextMinTimeStep variable is then used as the value for the ElcMinTimeStep (or "local" minimum resolvable time) for the next window. A smaller time-step for the next window may be used if a break point occurs in that window due to a piece-wise linear voltage source. The electrical nets are then iterated after the NetElcPreNewStep element returns until convergence. The NetElcNewStep element is then processed and it checks to see if the value of the ElcNextMinTimeStep variable decreased during convergence. If it did, the time-step for the next window is readjusted and the nets are realigned to the new local minimum resolvable time "grid". At this point, the new window is begun and the time parameters for the new window are copied into the current window time parameters, the NetElcPreNewStep and NetElcNewStep elements are scheduled at the end of the new window, and the ElcNextMinTimeStep variable is set to the full width of the new window so that the window size can grow.

When a time-step for a net is rejected, the new time-step chosen for that net may be smaller than the current local minimum resolvable time. If the new time-step is greater than or equal to the current local minimum resolvable time, then the net and its fanouts are re-scheduled to the largest integral multiple of the local minimum resolvable time that is not greater than the new time-step. If the new time-step is less than the current local minimum resolvable time, then the window parameters need to be recomputed and the nets realigned to the new time grid. It should be pointed out that simulators that cannot reject time-steps are not as good at controlling the error of the solution. The need to be able to backup time and re-index events puts strong demands on the scheduler. The cached indexing methods described in Chapter 2 are uniquely suited to simple time backup and re-indexing.

CHAPTER 5

Discrete Simulation

The implementation of the discrete levels of simulation in the SPLICE2 program is described in this chapter along with some background information. Logic simulation has been the subject of active research for many years [66]. Only those aspects of logic simulation that are different from more conventional logic simulation are dealt with in this chapter. Background on logic signals and models is presented in the first section. The details of logic simulation as implemented in the SPLICE2 program are described in the next section and the approach taken for the register transfer level (RTL) of simulation is presented in the last section.

5.1. Logic Signals and Models

Historically, logic simulation was first implemented with only two logic states [67], *true* and *false*, and the time behavior of the circuits modeled was *synchronous* so that no time delay information was used. The unknown state was added so that circuit hazard and race conditions could be detected during simulation [68, 69]. Open-collector logic gates presented a modeling problem for the logic simulator that was solved at first by adding a new gate, known as an *implicit wire-or*, that was inferred whenever the output of more than one open-collector device was connected to one wire. However, when tri-state gates and MOS switch logic became common, it was necessary to add more logic states to represent the possible drive characteristics at the outputs of the gates. As technology progressed and bi-directional switch simulation [70] became more widely used, more logic states were added to the basic three. Nine logic states are shown spread out on a plane in Figure 5.1 where the horizontal axis is the signal *level*, which corresponds to voltage, and the vertical axis is the signal *strength*, which corresponds to resistance, as has been described in Chapter 2.

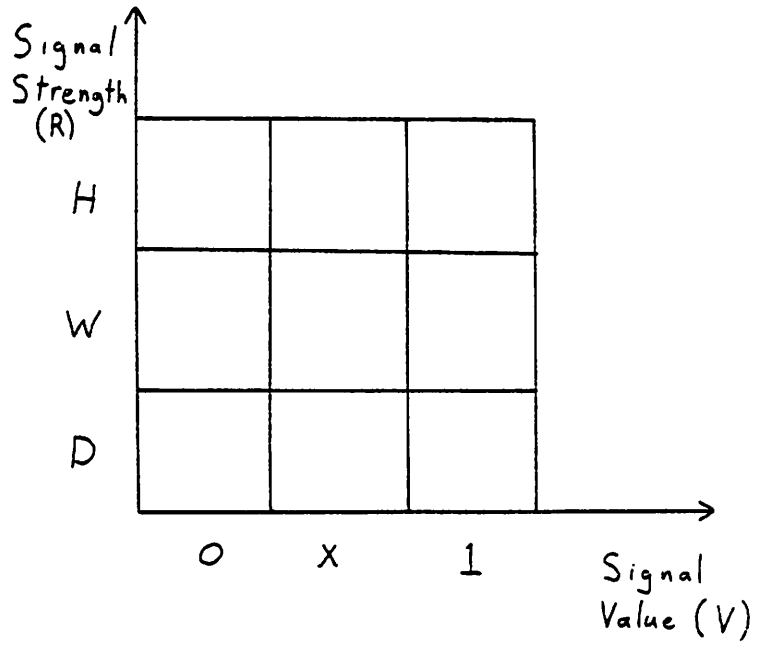


Figure 5.1 : Nine-State Logic System

5.1.1. How Many States Are Enough

The nine state logic system arises from the need to model transistors more accurately than fewer states will allow. Two examples are shown in Figure 5.2 that cannot be modeled adequately with only nine logic states. The first example, shown in Figure 5.2a, is a two-phase regenerative latch driving a bus and requires more than three resistance levels for proper simulation. The depletion transistor is used as a resistor to provide positive feedback from the output to the input of the buffer. When signal A is high, the output of the depletion transistor is overcome by the output of the transistor with input marked A and when signal A is low, the depletion transistor must prevail. Thus, three strengths are needed to model the contributions at node X. A fourth strength is needed for the output of the buffer so that when signal B is high, the buffer prevails over the on transistor and drives the bus value. The second example, shown in Figure 5.2b, is a fragment of pass-transistor logic that is probably the result of an error in design since the gate of transistor D is driven by a voltage that may be unsuitably low. The maximum voltage at C will be one threshold voltage drop below the voltage at B which in turn will be one threshold voltage drop below the voltage at A. Thus if A is already below the supply voltage, then signal C may be three threshold voltage drops below the supply voltage. If it is desired to detect this type of error, five logic voltage levels are needed. For proper simulation of a circuit containing both of these examples, 20 logic states would be needed as shown in Figure 5.2c. Note that since the threshold voltage is a function of the source voltage, the levels might be chosen to be nonuniform. The purpose here is not to propose 20 state simulation, but rather to illustrate how an electrical basis can be used to decide how many states are needed to simulate circuits of a given technology.

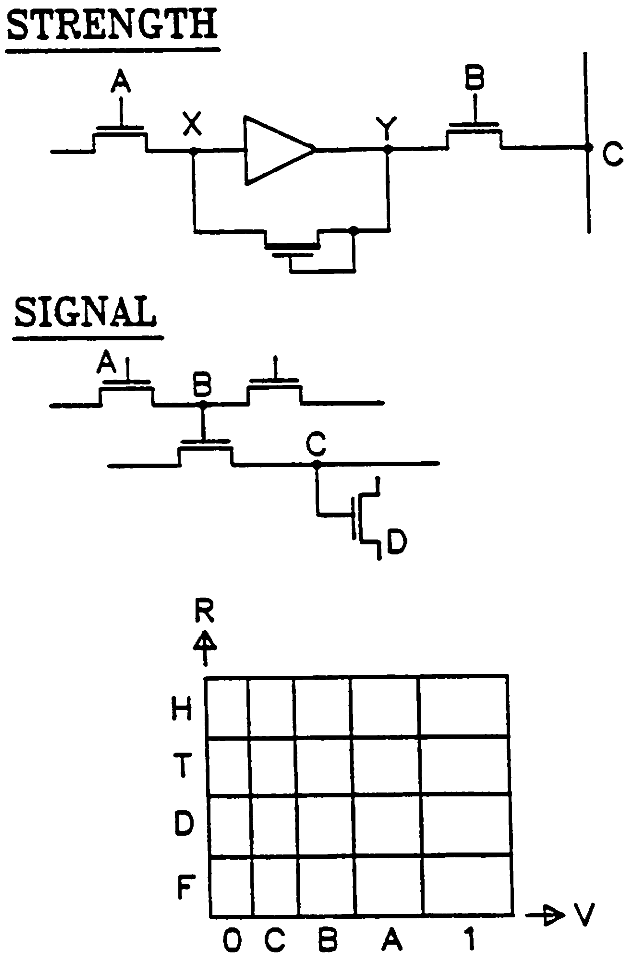


Figure 5.2 : Nine-State Counter-Examples

As stated in Chapter 2, signals are pairs of level and strength that are stored separately. This works out well for logic gates since outputs are typically computed from only the logic levels and the logic strengths are used separately to determine how to combine them. When tables are used to compute logic functions, their dimensions are reduced considerably by separation of level and strength, thus permitting more states for greater accuracy. Ideally, the number of logic signal levels and strengths, and their corresponding electrical levels should be defined in tables given by the user or stored in a technology library.

5.1.2. Logic-Electrical Interface

In the electrical domain, signals have values which are continuous, not discrete. The problem of interfacing logic and electrical simulation then is one of how to make an effective translation between the two levels so as to lose as little information as possible and to remain electrically consistent. The approach taken in the SPLICE1 program is illustrated in Figure 5.3a which shows a logic-to-voltage converter element. The logic-to-voltage element takes logic ones and zeros as input and converts them to specified voltages using a given rise and fall time. A logic-to-current converter element is used when a high impedance drive capability is needed. These two elements provide drive resistances of either zero or infinity. The DIANA program [28] uses a technique illustrated in Figure 5.3b, called the boolean controlled switch (BCS), that allows finite values of drive resistance to be used. The logic (or boolean) signal is used to control a switch that connects one of two voltage sources to the output node through one of two resistors. The choice of values for the voltages and resistances used to represent the logic signal is a modeling issue. In some cases, the small-signal Thevenin equivalent resistance may be appropriate and in others, the large-signal equivalent resistance may give a better fit. The signals for the logic-to-voltage, logic-to-current, and boolean controlled switch looking into the output terminal are shown in Figure 5.4 as trajectories in the resistance-voltage plane. The logic-to-voltage (LTV) and logic-to-current (LTI) converters are shown schematically as lines at zero and infinite resistance respectively that ramp smoothly from the low voltage to the high voltage. The boolean controlled switch (BCS) is shown as a pair of points and switching is discontinuous between them. The boolean controlled switch requires a grounded capacitor at its output terminal to give a finite switching time and the trajectory then becomes a rectangle containing the two points.

To answer the question of how well these two methods represent logic in the electrical domain, the Thevenin equivalent to ground is plotted parametrically in Figure 5.5 for an n-channel MOS inverter with a pass transistor connected to its output. This data was obtained

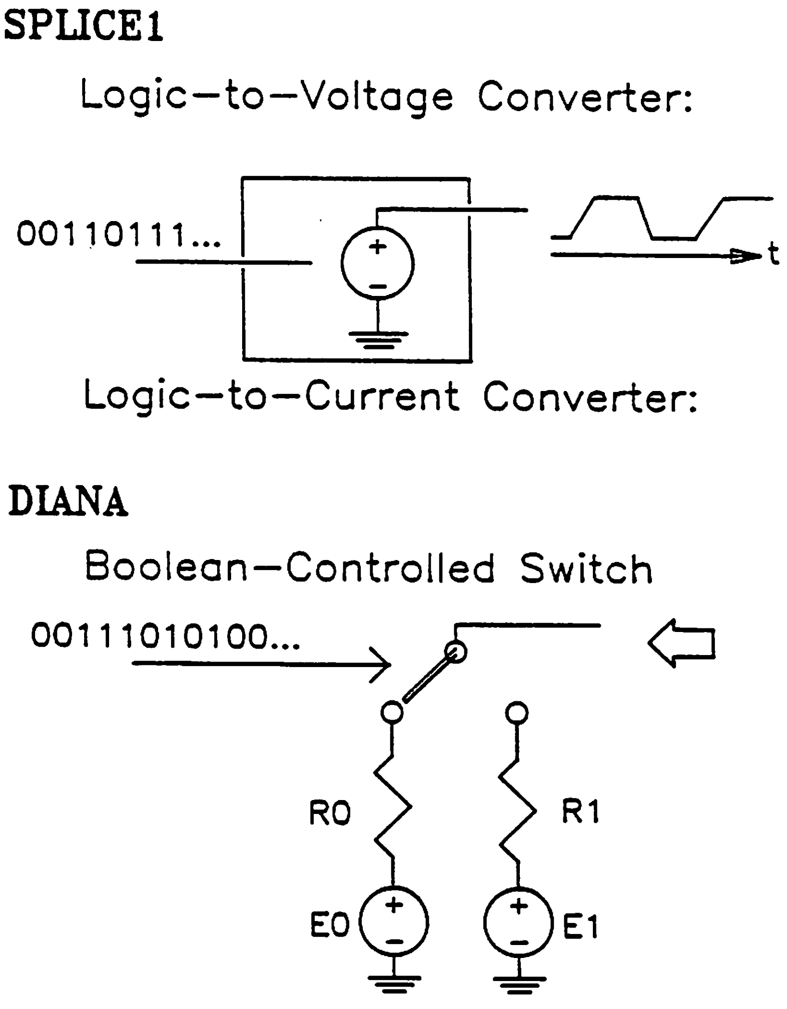

SPLICE1
Logic–to–Voltage Converter:
00110111...
+
-
t
Logic–to–Current Converter:
DIANA
Boolean–Controlled Switch
00111010100...
R0
R1
E0
E1
+
-
+
-

Figure 5.3 : Logic to Electrical Mapping

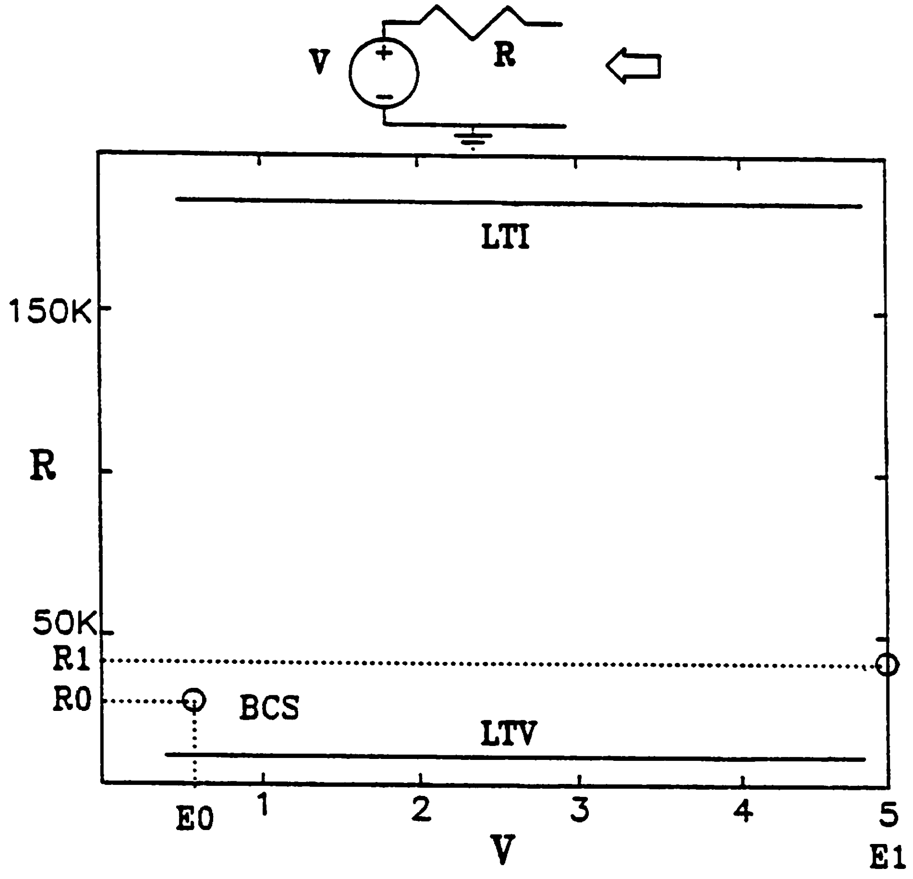


Figure 5.4 : Conversion Element Thevenin-Equivalents

by running the SPICE program and requesting a DC transfer function calculation while varying the input voltage to the inverter. The upper curve corresponds to the output through the pass transistor and can be seen to be a constant resistance larger than that of the output node of the inverter, shown in the lower curve, until around 2V when the gate to source voltage becomes less than the threshold voltage and the channel begins to turn off. Note that both curves are quite nonlinear. These curves are appropriate in a DC situation or when the output voltage on the inverter is able to adjust rapidly to the changing input voltage. At the other extreme, the output voltage is not allowed to change at all while the input voltage switches completely. This situation corresponds to an infinite capacitance at the output, and is illustrated in Figure 5.6 for the output of the inverter. The trajectory now splits, showing hysteresis that depends on the direction of the input voltage change. The upper curve results from the input voltage switching off, thus causing the drive transistor to switch completely off and allowing the depletion pull-up transistor to charge the capacitor. The lower curve results from the input voltage switching on and is the composite of the load curve and the driver curve. Note that all of the possible trajectories for the Thevenin output contribution lie inside the pair of curves shown in Figure 5.6 assuming that there are no bootstrap effects that cause the voltage to exceed the power supplies.

A boolean controlled switch can follow only a rectagonal path that represents a compromise between the "fast" and "slow" trajectories. To answer the question of how well a boolean controlled switch models the inverter, the pair of points for a boolean controlled switch that were chosen to best fit the waveforms of a chain of these inverters, each of which drives a 10fF load, is shown in Figure 5.6. The resistance values calculated were 18KΩ and 35KΩ The grounded capacitance at one of the inverter outputs was then increased to 200fF without readjusting the boolean controlled switch model. This resulted in a 40% error in the delay through that stage [5]. This implies that more than two states are required to describe the output contribution of the inverter adequately over this range of values. Further, the output contribution of the inverter depends not only on the input voltage but also on the present voltage at the output node.

These results suggest that in order to make discrete simulation technology independent, more levels than just 0 and 1 are needed in voltage and a number of strengths as well. Further, the interaction of elements and wires via signals should be based on electrical algorithms. Some work has been performed in the SPLICE2 program on an electrically-based discrete simulation method called ELOGIC (Electrical LOGIC simulation). The results have been encouraging, but more work is needed.

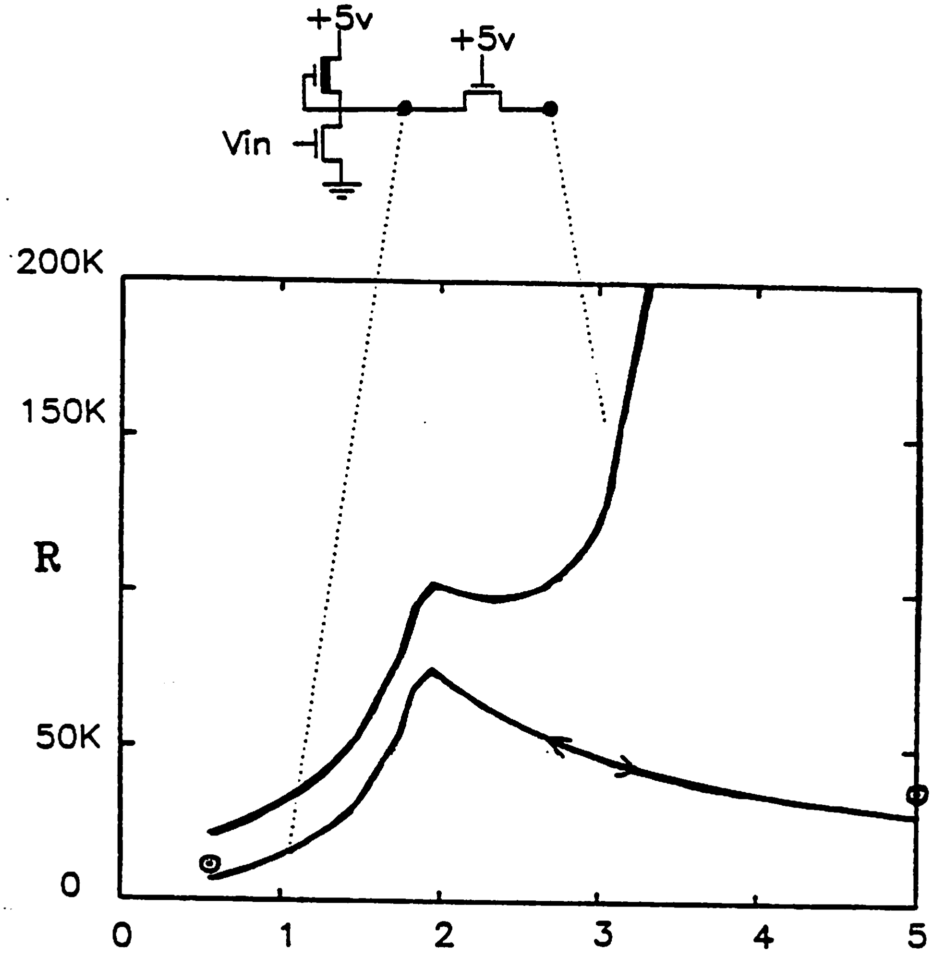


Figure 5.5 : Thevenin DC Curves for Inverter

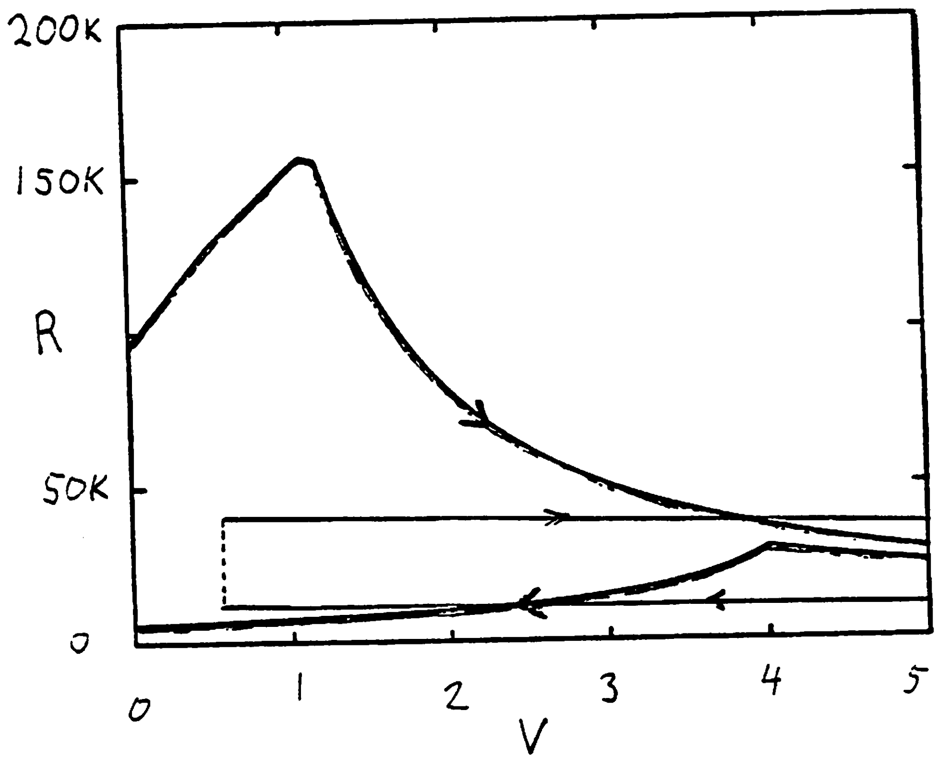


Figure 5.5: Thevenin Hysteresis for C=∞

5.1.3. Logic Delay Models

Logic simulators that allow gates to have assignable delays usually use inertial delay models [1]. This means that only one logic state change can propagate through a gate during a signal transition period and the output of the gate must attain its new value before another input change can be accepted. If the input changes before the output reaches its new value, this is said to be a *spike*. Spikes are a result of the presence of topological circuit *hazards*. Spikes may or may not be allowed to propagate to fanout gates depending on the particular logic simulator implementation. However, the meaning of spikes changes for many-valued logic simulation where there are more than two logic levels that represent legal states of the logic system. In this situation, it is likely that the inputs will change before the outputs finish changing under circumstances that are electrically meaningful.

Figure 5.7 shows a pair of inverters that have different rise and fall times. The falling input waveform to the second inverter can pass through several states on its way from high to low before the output of the second inverter has responded. The inertial delay model has been extended [5] to allow proper modeling of signal delays in this situation. All of the states between the present output state and the eventual output state must be visited during the transition, which makes sense physically. This rule implies that each of the logic level and logic strength may not change value by more than ±1. Spikes, or "runt" pulses, can still be detected as an output change that passes one logic threshold twice without passing the next one. If the user requests spike detection, net models can check for this case or other cases easily.

5.2. Logic Implementation in SPLICE2

The present implementation of logic simulation in the SPLICE2 program is a unidirectional logic gate capability using six logic levels and five logic strengths. The switch simula-

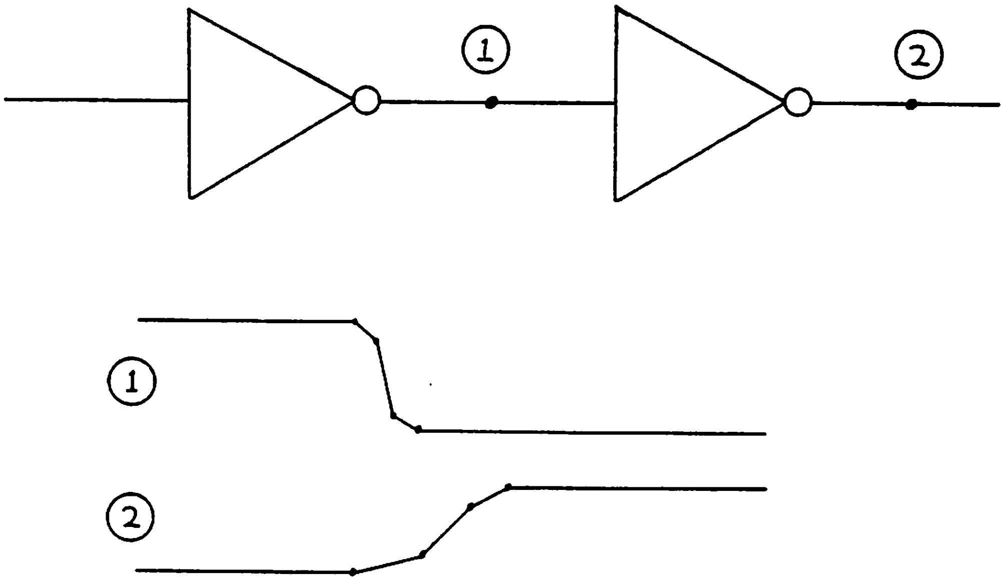


Figure 5.7 : Inertial Delays and Multiple States

tion capability from the SPLICE1 program [3] has not yet been incorporated in the SPLICE2 program. The six logic levels are:

1	True or high.
F	Falling.
X	Unknown or error condition.
IX	Initial unknown.
R	Rising.
0	False or low.

and the five logic strengths are:

S	Input source.
D	Driving output.
W	Weak output.
H	High impedance.
IH	Initial high impedance.

Note that using the mixed-mode simulation concept for a signal that separates the signal level from the signal strength allows a large number of states without the need for large tables when compared to a more conventional but equivalent approach. To represent all of the possible states above using a conventional approach would require 30 logic states. A table-driven element routine that has two inputs and uses a two-dimensional table would need 900 entries in its table. By separating out the strengths, the table using the logic levels listed above only requires 36 entries. The signal strengths are used by the logic net model when deciding how to combine multiple fanin signals.

Logic gates are implemented in the SPLICE2 program using table-driven models that can be found in Appendix A in the files "loggates.mod" and "logtables.c". The present logic tables do not actually produce the rising (R) or falling (F) logic levels on output and treat them as 0 and 1 respectively on input. The rising and falling states are intended for use with more advanced hazard detection and waveform modeling. By choosing whether or not to use those two logic levels, the simulation can be made more or less precise at the expense of CPU overhead. The user also has a choice of tables that treat the initial unknown level (IX) differently. The strict set of tables, which are the default, treat the initial unknown level in the same way as the normal unknown level. The other set allows inputs that are at the initial unknown level to be treated as if they were missing, or ignored [43]. This allows configurations of gates such as shown in Figure 5.8 to be initialized to some state where, with the strict interpretation of the initial unknown level, it would not be possible to initialize the outputs. The initial unknown level is useful also for detecting which signals in a circuit were never set to a valid state during the course of simulation.

The tables for the AND logic gate are given here as an example. For the complete set of tables, see Appendix A.

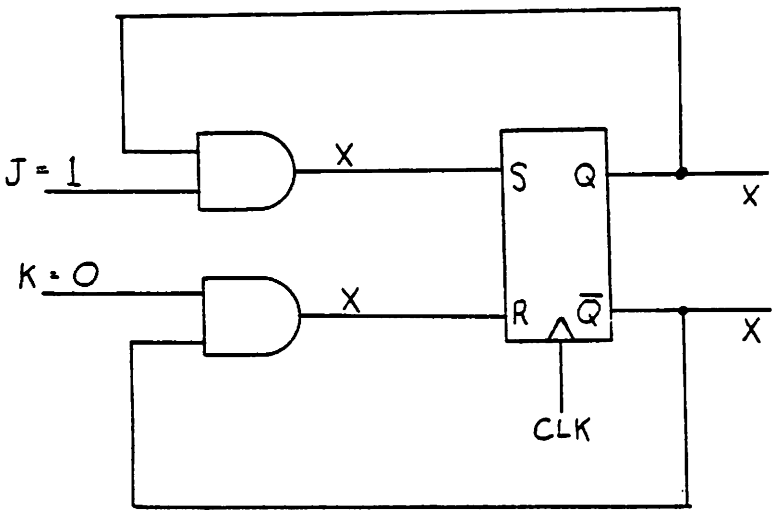


Figure 5.8 : Logic Example for Initial Unknown

```
/*
 * This version of the tables ignores the R and F states on input
 *  and treats them as 0 and 1 respectively.
 */

/*
 * Tables for AND, OR, and XOR logic functions.
 *  NAND, NOR and XNOR are constructed by using a post invert step.
 */

LogLevType   LogAndTable[ LOG_NUM_LEVELS ] [ LOG_NUM_LEVELS ] = {
/*           LOG_0    LOG_1    LOG_X    LOG_IX   LOG_R    LOG_F */
/*           ==================================================== */
/* LOG_0  */ { LOG_0, LOG_0,   LOG_0,   LOG_0,   LOG_0,   LOG_0,  },
/* LOG_1  */ { LOG_0, LOG_1,   LOG_X,   LOG_IX,  LOG_0,   LOG_1,  },
/* LOG_X  */ { LOG_0, LOG_X,   LOG_X,   LOG_IX,  LOG_0,   LOG_X,  },
/* LOG_IX */ { LOG_0, LOG_IX,  LOG_IX,  LOG_IX,  LOG_0,   LOG_IX, },
/* LOG_R  */ { LOG_0, LOG_0,   LOG_0,   LOG_0,   LOG_0,   LOG_0,  },
/* LOG_F  */ { LOG_0, LOG_1,   LOG_X,   LOG_IX,  LOG_0,   LOG_1,  },
}       ;
```

```
/*
 * These logic tables are for use when the user wants to be able
 *  to allow the logic gate to ignore the initial unknowns that are
 *  on an input.  i.e. LOG_IX acts as a "missing" input.
 *  The suffix "IX" is appended to the table names to mean Ignore X.
 */

LogLevType    LogAndTableIX[ LOG_NUM_LEVELS ] [ LOG_NUM_LEVELS ] = {
/*            LOG_0    LOG_1    LOG_X    LOG_IX   LOG_R    LOG_F  */
/*            ===================================================== */
/* LOG_0 */   { LOG_0, LOG_0,   LOG_0,   LOG_0,   LOG_0,   LOG_0,   },
/* LOG_1 */   { LOG_0, LOG_1,   LOG_X,   LOG_1,   LOG_0,   LOG_1,   },
/* LOG_X */   { LOG_0, LOG_X,   LOG_X,   LOG_X,   LOG_0,   LOG_X,   },
/* LOG_IX */  { LOG_0, LOG_1,   LOG_X,   LOG_IX,  LOG_0,   LOG_1,   },
/* LOG_R */   { LOG_0, LOG_0,   LOG_0,   LOG_0,   LOG_0,   LOG_0,   },
/* LOG_F */   { LOG_0, LOG_1,   LOG_X,   LOG_1,   LOG_0,   LOG_1,   },
}       ;
```

The AND, OR, and XOR functions are implemented using two-dimensional tables since multiple lookups can be used for these functions to compute larger fanin cases i.e., these functions are associative. The NAND, NOR, and XNOR functions are implemented by using the corresponding positive logic table and then adding a post-invert table lookup. This is used even for the positive logic functions to get the technology-specific drive characteristics of the gate. The gate technology drive type is specified as a parameter on the gate model.

```
/*
 * These tables are used to represent the output drive characteristics
 *  of different logic families.  A parameter on the gate model selects
 *  which of these output types the gate has.
 */

Log_Sig       LogOpenCollectorOutTable[ LOG_NUM_LEVELS ] = {
/* LOG_0 */   { LOG_0,       LOG_D  },
/* LOG_1 */   { LOG_1,       LOG_H  },
/* LOG_X */   { LOG_X,       LOG_D  },
/* LOG_IX */  { LOG_IX,      LOG_IH },
/* LOG_R */   { LOG_R,       LOG_H  },
/* LOG_F */   { LOG_F,       LOG_D  },
}       ;

Log_Sig       LogInvOpenCollectorOutTable[ LOG_NUM_LEVELS ] = {
/* LOG_0 */   { LOG_1,       LOG_H  },
/* LOG_1 */   { LOG_0,       LOG_D  },
/* LOG_X */   { LOG_X,       LOG_D  },
/* LOG_IX */  { LOG_IX,      LOG_IH },
/* LOG_R */   { LOG_F,       LOG_D  },
/* LOG_F */   { LOG_R,       LOG_H  },
}       ;

Log_Sig       LogNmosOutTable[ LOG_NUM_LEVELS ] = {
/* LOG_0 */   { LOG_0,      LOG_D },
/* LOG_1 */   { LOG_1,      LOG_W },
/* LOG_X */   { LOG_X,      LOG_D },
/* LOG_IX */  { LOG_IX,     LOG_IH },
/* LOG_R */   { LOG_R,      LOG_W },
/* LOG_F */   { LOG_F,      LOG_D },
}       ;

Log_Sig       LogInvNmosOutTable[ LOG_NUM_LEVELS ] = {
/* LOG_0 */   { LOG_1,      LOG_W },
/* LOG_1 */   { LOG_0,      LOG_D },
/* LOG_X */   { LOG_X,      LOG_D },
/* LOG_IX */  { LOG_IX,     LOG_IH },
/* LOG_R */   { LOG_F,      LOG_D },
/* LOG_F */   { LOG_R,      LOG_W },
}       ;

Log_Sig       LogCmosOutTable[ LOG_NUM_LEVELS ] = {
/* LOG_0 */   { LOG_0,      LOG_D },
/* LOG_1 */   { LOG_1,      LOG_D },
/* LOG_X */   { LOG_X,      LOG_D },
/* LOG_IX */  { LOG_IX,     LOG_IH },
/* LOG_R */   { LOG_R,      LOG_D },
/* LOG_F */   { LOG_F,      LOG_D },
}       ;

Log_Sig       LogInvCmosOutTable[ LOG_NUM_LEVELS ] = {
/* LOG_0 */   { LOG_1,      LOG_D },
/* LOG_1 */   { LOG_0,      LOG_D },
/* LOG_X */   { LOG_X,      LOG_D },
/* LOG_IX */  { LOG_IX,     LOG_IH },
/* LOG_R */   { LOG_F,      LOG_D },
/* LOG_F */   { LOG_R,      LOG_D },
}       ;
```

5.3. Register-Transfer Level

Signals at the register-transfer level of simulation are represented with a binary value for both the signal level and signal strength. Further, the signals for more than one actual wire can be clustered into one logical signal. Thus, the signal is stored as a pair of arrays of bits, one for the signal level on each wire and another for the signal strength on each wire. This separation of signal level and strength is consistent with the representation of signals at all simulation levels. Further, it is very efficient for the simulation of certain types of register level models since the arithmetic and logical bit operations defined in the C programming language can be used directly.

The signal strength bit-array is necessary to be able to model wires or busses with more than one element driving them. The meaning of the signal strength at this level reduces to whether or not the element producing the signal is attempting to drive the bus or not. This allows a fast check that all of the inputs to an element are *valid* by testing that all of the bits are true. Also, those models for which it is possible to propagate valid outputs for some of the output bits even in the presence of some invalid inputs can do so. An example of this is a register latch since it only needs to invalidate the output bits for which there are invalid inputs. Other models, such as multiplexors, might invalidate all of their output bits.

The unknown logic level is not representable directly at the register level since there are only two possibilities, 0 and 1. However, since the meaning of the signal strength at the register level is whether the signal is driven or not i.e., valid, the unknown logic level can be translated to the register level by invalidating the register signal strength. There is still the problem of telling the difference between a truly undriven signal and an unknown one and also how to produce a logic unknown level from a register model. Both of these problems can be overcome if, when the register signal strength indicates that the signal is not driven, the signal level is used to indicate the difference between unknown and undriven. Thus, the following map is used to translate from logic signals to register signals:

```
struct  {
        unsigned char   level, stren;
}       LogElmLogToRTL[ LOG_NUM_LEVELS ] [ LOG_NUM_STRENGTHS ] = {
/*              LOG_IH          LOG_H           LOG_W           LOG_D           LOG_S  */
/*              ==============================================================  */
/* LOG_0  */  { {0, 0},         {0, 0},         {0, 1},         {0, 1},         {0, 1}   },
/* LOG_1  */  { {0, 0},         {0, 0},         {1, 1},         {1, 1},         {1, 1}   },
/* LOG_X  */  { {1, 0},         {1, 0},         {1, 0},         {1, 0},         {1, 0}   },
/* LOG_IX */  { {1, 0},         {1, 0},         {1, 0},         {1, 0},         {1, 0}   },
/* LOG_R  */  { {0, 0},         {0, 0},         {0, 1},         {0, 1},         {0, 1}   },
/* LOG_F  */  { {0, 0},         {0, 0},         {1, 1},         {1, 1},         {1, 1}   },
}       ;
```

From the table, it can be seen that the unknown level causes the strength of the register signal to be undriven. In translating from the register simulation level to the logic simulation level, the signal strength translates to a driving (D) or high impedance (H) logic strength when the register strength is 1 or 0, respectively. The register signal level translates directly to the corresponding logic level when the register signal strength is 1 and translates to logic level unknown (X) and false (0) when the register signal level and strength are (1,0) and (0,0) respectively. The mapping from register to logic signals is shown in the following table.

	Strength	
Level	1	0
1	1, D	X, H
0	0, D	0, H

The register level net model computes its new signal state by evaluating all of its fanins and if none of them have a driving strength, sets its signal to (0,0). If some of the fanins have a driving strength and all of them have the same signal level, the output is set to that level with a strength of 1. If there is a conflict between two driving signals, the signal is set to (1,0) to indicate the conflict and so that logic fanouts will get a logic level unknown. The following program fragment from the file *rtlnet.c* illustrates how this is achieved for a 32 bit signal. If the signal bit-array size does not fit into one word of storage, the algorithm must be repeated for each signal word.

```
/* For now, assume 32 bits or less. */
strength = level = invalid = 0;
for ( i = 0; i < numFanins; i++ ) {
     invalid  |= ( strength & sig[i].stren & (level ^ sig[i].level) )
               | ( sig[i].level & ~(sig[i].stren) );
     level    |= sig[i].level & sig[i].stren;
     strength |= sig[i].stren;
}
/*
 * Invalidate any strengths for bits driven both directions.
 */
level  |= invalid;
strength  &= ~invalid;
```

At the end of the algorithm, the level and strength variables contain the proper values to indicate contention, if necessary, and the driven value otherwise.

The following program fragment contains the data structure declaration and model routine for a 74181 arithmetic logic unit (ALU) for use in the SPLICE2 program. A description of the important parts of the model appears at the end of the fragment.

```
/* BEGIN_HEADER_DECLARATION */
struct  ElmRTL74181          {       /* ATTRIBUTES = ELMOBJECT */
        struct  ElmBasic     bas;
        SigElmRegister       lastValue;
}       ;
typedef struct  ElmRTL74181  ElmRTL74181;
struct  ModRTL74181          {       /* ATTRIBUTES = MODOBJECT */
        struct  ModBasic     bas;
/* PARAMETERS */
/* NETS f       REGISTER OUTPUT
        s       REGISTER INPUT
        m       REGISTER INPUT
        a       REGISTER INPUT
        b       REGISTER INPUT */
}       ;
typedef struct  ModRTL74181  ModRTL74181;
/* END_HEADER_DECLARATION */

/*
 * ElmRTL74181 - 7400 series ALU.
 */
ELM_EVAL_DECLARE(RTL74181)
{
   ElmNetBasic **lis;
   SigStruct    aSig, bSig, sSig, mSig;
   BitArray     level,  strength;

   /*
    * Evaluate the input nets.
    */
   DASSERT( elm->bas.fan.numIn == 4 );
   DASSERT( elm->bas.fan.numOut == 1 );
   lis = elm->bas.fan.in.lis;
   NET_EVAL_FUNC( lis[0], SIG_NET_REGISTER, & sSig, SplCurTime );
   NET_EVAL_FUNC( lis[1], SIG_NET_REGISTER, & mSig, SplCurTime );
   NET_EVAL_FUNC( lis[2], SIG_NET_REGISTER, & aSig, SplCurTime );
   NET_EVAL_FUNC( lis[3], SIG_NET_REGISTER, & bSig, SplCurTime );
   /*
    * Check for validity of input drive states.  Propagate unknown
    *  if no good.
    */
   strength.bits = sSig.netRTL.cur.stren.bits
           & aSig.netRTL.cur.stren.bits & bSig.netRTL.cur.stren.bits;
   if ( strength.bits != 0xf | mSig.netRTL.cur.stren.bits != 1 ) {
        level.bits    = 0xf;
        strength.bits = 0;
   }
```

```
    else {
        switch ((mSig.netRTL.cur.level.bits<<4) + sSig.netRTL.cur.level.bits ) {
#define ALUFUNC(m,s3,s2,s1,s0) m * 0x10 + s3 * 0x8 + s2 * 0x4 + s1 * 0x2 + s0
#           define F level.bits
#           define A aSig.netRTL.cur.level.bits
#           define B bSig.netRTL.cur.level.bits
#           define carry 0
            /*
             * Calculate new function value.
             */
            case ALUFUNC(0,0,0,0,0):  F = A + carry;                          break;
            case ALUFUNC(0,0,0,0,1):  F = (A | B) + carry;                    break;
            case ALUFUNC(0,0,0,1,0):  F = (A | ~ B) + carry;                  break;
            case ALUFUNC(0,0,0,1,1):  F = -1 + carry;                         break;
            case ALUFUNC(0,0,1,0,0):  F = A + (A & ~ B) + carry;              break;
            case ALUFUNC(0,0,1,0,1):  F = (A | B) + (A & ~ B) + carry;        break;
            case ALUFUNC(0,0,1,1,0):  F = A - B - 1 + carry;                  break;
            case ALUFUNC(0,0,1,1,1):  F = (A & ~ B) - 1 + carry;              break;
            case ALUFUNC(0,1,0,0,0):  F = A + (A & B) + carry;                break;
            case ALUFUNC(0,1,0,0,1):  F = A + B + carry;                      break;
            case ALUFUNC(0,1,0,1,0):  F = (A | ~ B) + (A & B) + carry;        break;
            case ALUFUNC(0,1,0,1,1):  F = (A & B) - 1 + carry;                break;
            case ALUFUNC(0,1,1,0,0):  F = A + A + carry;                      break;
            case ALUFUNC(0,1,1,0,1):  F = (A | B) + A + carry;                break;
            case ALUFUNC(0,1,1,1,0):  F = (A | ~ B) + A + carry;              break;
            case ALUFUNC(0,1,1,1,1):  F = A - 1 + carry;                      break;
            /* Logic functions */
            case ALUFUNC(1,0,0,0,0):  F = ~ A;                                break;
            case ALUFUNC(1,0,0,0,1):  F = ~ (A | B);                          break;
            case ALUFUNC(1,0,0,1,0):  F = ~ A & B;                            break;
            case ALUFUNC(1,0,0,1,1):  F = 0;                                  break;
            case ALUFUNC(1,0,1,0,0):  F = ~ (A & B);                          break;
            case ALUFUNC(1,0,1,0,1):  F = ~ B;                                break;
            case ALUFUNC(1,0,1,1,0):  F = A ^  B;                             break;
            case ALUFUNC(1,0,1,1,1):  F = (A & ~ B);                          break;
            case ALUFUNC(1,1,0,0,0):  F = ~ A | B;                            break;
            case ALUFUNC(1,1,0,0,1):  F = ~ (A ^  B);                         break;
            case ALUFUNC(1,1,0,1,0):  F = B;                                  break;
            case ALUFUNC(1,1,0,1,1):  F = A & B;                              break;
            case ALUFUNC(1,1,1,0,0):  F = 1;                                  break;
            case ALUFUNC(1,1,1,0,1):  F = A | ~ B;                            break;
            case ALUFUNC(1,1,1,1,0):  F = A | B;                              break;
            case ALUFUNC(1,1,1,1,1):  F = A;                                  break;
            default:
                Error( INTERNAL, "ElmRTL74181: Impossible state." );
                break;
        }
    }
    result->elmRTL.sig.level      = level;
    result->elmRTL.sig.stren      = strength;
    result->elmRTL.sig.time.val   = SplCurTime;
    result->elmRTL.numBits        = 4;
    return( SIG_ELM_REGISTER );
```

```
101  }
```

The model for the 74181 ALU shown above is written in the C programming language using some text substitution macros to improve readability. Macro definitions could be used to shield a designer from the specifics the underlying syntax of the C programming language. For example, the logical operations on lines 80-84 could be made more familiar to a designer by use of the macros AND, OR, NOT, and XOR to indicate the corresponding logical operations:

```
80      case ALUFUNC(1,0,1,0,1):   F = NOT( B );                break;
81      case ALUFUNC(1,0,1,1,0):   F = XOR( A, B );             break;
82      case ALUFUNC(1,0,1,1,1):   F = AND( A, NOT(B) );        break;
83      case ALUFUNC(1,1,0,0,0):   F = OR( NOT(A), B );         break;
84      case ALUFUNC(1,1,0,0,1):   F = NOT( XOR(A, B) );        break;
```

This process could be carried further by supplying a preprocessor program that reads user-supplied model functions and translates them into C functions.

Specifying a new register level model involves declaring two data structures and providing one function for use in evaluating the model. For the 74181 ALU model above, the data structure declarations are between line 1 and line 17. These lines are used to delimit the *element* and *model* data structure declarations and signal that information about the model will be passed in the comments. One element data structure is allocated and associated with each instance of the ALU model. One model data structure is allocated and associated with each model declaration in the input data. On lines 3 and 8, the basic element and model information is included for these data structures. The signal *lastValue* that is declared on line 4 is a storage location used by the model to store the last value output for each instance of the ALU model. There are no parameters for either the element or model data structures. If there were any, they would appear as members of the structure after the keyword *PARAMETERS* such as shown on line 9. The topological information about fanins and fanouts is declared on lines 10-14 by using the keyword *NETS* and then listing all of the wires expected for the element. Three pieces of information are provided for each wire: the name of the input or output, the level expected by it (which is one of REGISTER, LOGIC, and

CIRCUIT), and the direction of the signal flow (which is one of INPUT, OUTPUT, and INANDOUT). On line 22, the evaluation function for this element is declared using a macro expansion in order to guarantee that all of the types and arguments for the function are declared correctly. The macro prepends the prefix "Ele" (which means element evaluation function) to the model name and indicates that the return value of the function is an enumerated type for the signal level called "Sig_Level". Three arguments are automatically declared. The first is a pointer to the element instance that is to be evaluated with this routine (called "elm"), the second is a pointer to the net that is requesting the evaluation (called "net"), and the third is a pointer to a buffer for the resulting signal (called "result"). Signal buffers that are unions of the possible signal types are declared on line 25 for use when evaluating the input nets. Bit-arrays are declared on line 26 to hold the resulting output level and strength that are computed for the model. The input buffers are filled in lines 34-37 by invoking a each input net's evaluation function. In lines 42-46, the validity of the input signal strengths is checked, and if any inputs are invalid, the output for the element is set to invalid. Otherwise, a function code for the ALU is computed on line 49 and used in the switch on lines 58-93 to decide which function is being requested of the ALU. The purpose of the definitions on lines 51-54 is to abbreviate the program code and allow the form to appear the same as that which appears in the tables of the 74181 part data sheet. The computed signal is stored into the result buffer provided by the calling routine on lines 96-100. It is interesting to note how the ALU functions in the switch are implemented using the intrinsic arithmetic and logic operations of the C programming language. Thus on a VAX, the model is capable of representing an ALU of up to 32 bits in width with exactly the same CPU time costs.

Register-transfer level simulation such as described above is not the same as a behavioral level of simulation. However, since the models for the elements are written in a high-level programming language (C), a behavioral level of simulation could be layered onto the register level of simulation. The key is that the primitives of the simulator are available through macro calls and that the high-level models can communicate with the lower-level models by using the established coercions. However, inside the model, the model code can perform any sort of algorithm that can be implemented in a programming language. As an example, say that it is desired to simulate the behavior of a packet switched data communication network protocol with varying packet sizes. The bit-array abstraction allows arbitrary data types to be layered on top of the existing signal data types. The packet header could be defined with the appropriate number of bits and transmitted along with the data on wires specified to the simulator. In this scenario, the mixed-level simulator merely provides the framework within which the high-level simulation is embedded. At the present time, behavioral level simulation is not supported by the SPLICE2 program.

CHAPTER 6

Examples

The purpose of this chapter is to characterize and demonstrate the use of different simulation methods in the SPLICE2 mixed-mode simulation program. Program statistics and other measurements from the SPLICE1 and SPICE2 programs are presented, where appropriate, for the purpose of comparison. The example circuits used are described briefly in the next section and measurements are presented in the section following. An excellent set of measurements for the SPLICE1 program which covers a wide range of circuit types is given in [3].

6.1. Description of Example Circuits

Figure 6.1 contains a list of abbreviated names for the example circuits along with a description of each one. The names for each example are constructed from a short root that is descriptive of the experiment and a suffix indicating information such as the levels of simulation used and types of logic gates used to implement the logic. The simulation levels are identified with "R", "L", "E", and "E2", for register, logic, iterated timing analysis implemented in the SPLICE2 program in a way similar to the way it is implemented in the SPLICE1 program (ITA), and iterated timing analysis using the variable window mechanism described in Chapter 4 (ITA2), respectively. Further, some of the examples are a series of logic circuits taken from the TTL Data Book [71] and implemented with elementary logic gates. For these examples, there is also a letter "A" or "O" that stands for whether or not AND gates were used. The letter "A" implies that AND and NAND gates were allowed where MOS transistors are connected in series and that the circuits were implemented as close to the TTL gate descriptions as possible. The letter "O" implies that AND and NAND gates were implemented using OR and NOR gates with inverters to provide logical equivalence and have only transistors that have at least one of the drain or source nodes connected to a power supply node. The all NOR gate circuits provide examples for iterated timing analysis that do not have strong bilateral coupling between nets and are unidirectional while the NAND gate circuits have strong bilateral coupling due to the bidirectional transistors. As an example, "dffOL" means a description of a 7474 D-type flip-flop implemented with logic gates that are all OR and NOR gates.

Each measurement reported in this chapter has an abbreviated name for use in tables that is shown in Figure 6.2 along with a short description of the meaning of that measurement. Also, the non-proprietary circuit example input files can be found in Appendix B along with files containing the raw measurements. Hierarchical run-time execution profile statistics [72] are also given in Appendix B for selected examples. For ease of reference, the names of the files in which the raw results can be found in Appendix B are given for each example in Figure 6.3.

Experiment	Description
aluRL	74181 RTL ALU with three 74163 logic counters.
biglogL	Sixteen bit counter made from four 4-bit counters.
cdeAE	Counter-Decoder-Encoder all electrical (NAND gates).
cdeAE2	Counter-Decoder-Encoder all ITA-2 (NAND gates).
cdeL	Counter-Decoder-Encoder with logic gates.
cdeLE	Counter-Decoder-Encoder with logic counter, rest electrical.
cdeOE	Counter-Decoder-Encoder all electrical (NOR gates).
cdeOE2	Counter-Decoder-Encoder all ITA-2 (NOR gates).
countAE	Four bit binary counter (74163) all electrical.
countAE2	Four bit binary counter (74163) all ITA-2 (NAND gates).
countL	Four bit binary counter (74163) all logic gates.
countOE	Four bit binary counter (74163) all electrical (NOR gates).
countOE2	Four bit binary counter (74163) all ITA-2 (NOR gates).
dffAE	D type flip flop. ITA (NAND gates).
dffAE2	D type flip flop. ITA-2 (NAND gates).
dffL	D type flip flop. Logic gates.
dffOE	D type flip flop. ITA (NOR gates).
dffOE2	D type flip flop. ITA-2 (NOR gates).
nandAE	Two-input nand gate. ITA (NAND gate implementation).
nandAE2	Two-input nand gate. ITA-2 (NAND gate implementation).
nandL	Two-input nand gate. Logic gate.
nandOE	Two-input nand gate. ITA (NOR gate implementation).

Figure 6.1 : Example Circuit Descriptions

Measurement	Description
COMP_HR	Composite Scheduler Hit Ratio
CPU_ANAL	CPU Analysis User Time
CPU_ANAL_SYS	CPU Analysis System Time
CPU_READ	CPU Readin and Setup User Time
CPU_READ_SYS	CPU Readin and Setup System Time
ELC_BUF	Number of Buffers Used for ITA
FI_MEAN	Mean Number of Fanins to Nets
FO_MEAN	Mean Number of Fanouts from Nets
MEM_BLT	Memory (bytes) used by BLT
MEM_NET	Net Memory Exclusive of BLT
NEN	Number of Electrical Nets
NEXE	Number of Net-Executions
NLN	Number of Logic Nets
NRESCH	Number of Re-Schedules
NRN	Number of Register Nets
NSCHED	Number of Schedules
NSCH_HIT	Number of SchedCache Hits
NSCH_MI	Number of SchedCache Misses
NSCH_TT_HIT	Number of SchedAtThisTimeCache Hits
NSCH_TT_MI	Number of SchedAtThisTimeCache Misses
NTBU	Number of Time Backups
NUNSCH	Number of Un-Schedules
N_ACC_S	Number of Accepted Solutions
N_ELMS	Number of Non-Net Elements Allocated
N_FL_S	Number Solutions Flushed
N_IT_REJ	Number of Iteration Limit Rejections
N_MODELS	Number of Models Allocated
N_NET_ELMS	Number of Net Elements Allocated
N_PR_S	Number of Solutions Written to Output
N_REJ_S	Number of Rejected Solutions
N_SYNC	Number of ITA Synchronizations
N_TOT_EV	Total Number of Net and Element Evaluations
N_WINDOW	Number of ITA Windows Used
SCH_HR	SchedCache Hit Ratio
SCH_TT_HR	SchedAtThisTimeCache Hit Ratio
TOT_ITER	Total Number of Iterations
USE_ITER	Number of 'Useful' Iterations

Figure 6.2 : Measurement Descriptions

Experiment	File Name
aluRL	rtl/testrtl.sp2.out
biglogL	logic/biglog.sp2.out
cdeAE	ita/nandgates/cde.sp2.out
cdeAE2	ita2/nandgates/cde.sp2.out
cdeL	logic/cde.sp2.out
cdeLE	mixed/cde.sp2.out
cdeOE	ita/norgates/cde.sp2.out
cdeOE2	ita2/norgates/cde.sp2.out
countAE	ita/nandgates/counter.sp2.out
countAE2	ita2/nandgates/counter.sp2.out
countL	logic/counter.sp2.out
countOE	ita/norgates/counter.sp2.out
countOE2	ita2/norgates/counter.sp2.out
dffAE	ita/nandgates/dff.sp2.out
dffAE2	ita2/nandgates/dff.sp2.out
dffL	logic/dff.sp2.out
dffOE	ita/norgates/dff.sp2.out
dffOE2	ita2/norgates/dff.sp2.out
nandAE	ita/nandgates/nand.sp2.out
nandAE2	ita2/nandgates/nand.sp2.out
nandL	logic/nand.sp2.out
nandOE	ita/norgates/nand.sp2.out

Figure 6.3 : Raw Result File Names

Some statistics about each circuit example are listed in Figure 6.4 and include the number of electrical, logic, and register nets along with the mean number of fanins and fanouts at each net. The examples that are completely logic simulation are seen to have one fanin to each net and about 2.3 fanouts from each net. The NOR-gate implementation of the circuits can be seen to have substantially more fanins and fanouts per net than the NAND-gate equivalent circuits. The electrical examples also have a large number of grounded capacitors which increase the number of fanins to each net.

6.2. Measurements

This section contains a summary of statistical data gathered on the circuit examples presented above. Where appropriate, comparisons are made with the SPLICE1 and SPICE2 programs. The CPU times required for the dc-transient simulation of the examples are given in Figure 6.5 along with values normalized to that of the simulation using the SPLICE2 pro-

Experiment	Levels	NEN	NLN	NRN	FI_MEAN	FO_MEAN
aluRL	RL	3	120	6	0.954	2.305
biglogL	L	3	237	0	0.975	2.354
cdeAE	E	453	0	0	8.818	4.695
cdeAE2	E2	453	0	0	8.818	4.695
cdeL	L	3	155	0	0.963	2.385
cdeLE	EL	291	63	0	7.363	4.242
cdeOE	E	555	0	0	12.465	7.142
cdeOE2	E2	555	0	0	12.465	7.142
countAE	E	171	0	0	8.468	4.491
countAE2	E2	171	0	0	8.468	4.491
countL	L	3	63	0	0.913	2.072
countOE	E	225	0	0	12.035	6.833
countOE2	E2	225	0	0	12.035	6.833
dffAE	E	23	0	0	4.607	2.357
dffAE2	E2	23	0	0	4.607	2.357
dffL	L	5	8	0	0.611	0.889
dffOE	E	35	0	0	8.925	4.950
dffOE2	E2	35	0	0	8.925	4.950
nandAE	E	9	0	0	1.750	0.750
nandAE2	E2	9	0	0	1.750	0.750
nandL	L	5	3	0	0.545	0.182
nandOE	E	11	0	0	3.786	1.929

Figure 6.4 : Example Circuit Characteristics

gram. Values which are marked "NA" imply that the simulation of that example with the corresponding simulator has not been done because it is not appropriate. Comparisons of ITA in the SPLICE2 program with the SPICE2 program show a CPU speedup of from about a factor of 10 to a factor of 100 increase in the simulation speed. Comparisons of ITA in the SPLICE2 program with the SPLICE1 program show a simulation speed that is roughly comparable. The ITA2 algorithm in the SPLICE2 program is seen in Figure 6.5 to require less time than that of the SPICE2 program, but significantly more time than that of ITA in the SPLICE2 program. This is due to two differences between the algorithms. First, the ITA2 algorithm allows very small time-steps to be taken depending on the estimate of the truncation error whereas the SPLICE1 program uses a fixed value for the minimum time-step. Second, the windowing mechanism prevents the ratio of time-step between slowly changing nets and quickly changing nets from being as large as it could be. Improvements to the windowing mechanism would likely improve the speed of the ITA2 method greatly. It can also be seen from Figure 6.5 that the run times for the "OR" gate implementations are worse for ITA2 versus ITA than the run times for the "AND" implementations. This is the result of a higher average fanout at each net as shown in Figure 6.4. Thus, since a non-converged net causes its fanouts to be scheduled in addition to itself, considerably more work is done when the average number of fanouts is higher. The numbers for fanin and fanout at a net in Figure 6.4 do not accentuate the differences between the "OR" and "AND" implementations since more of the fanouts in the "OR" implementation are voltage source nets.

The logic/electrical example labeled "cdeLE" is the same as the all electrical example labeled "cdeAE" except that the four-bit counter is replaced with a logic gate version. It can be seen from Figure 6.5 that the analysis time for the "cdeLE" example is about equal to the

Experiment	SPLICE2		SPLICE1		SPICE2	
	CPU	NORM	CPU	NORM	CPU	NORM
aluRL	92.1	1	NA	NA	NA	NA
biglogL	339	1	180	0.53	NA	NA
cdeAE	3183	1	2005	0.63	73562	23
cdeAE2	19946	6.3	"	"	"	"
cdeL	7.3	1	9.8	1.3	NA	NA
cdeLE	1765	1	1078	0.61	NA	NA
cdeOE	1211	1	1225	1.0	125681	104
cdeOE2	26695	22	"	"	"	"
countAE	1401	1	950	0.68	18714	13
countAE2	5816	4.2	"	"	"	"
countL	3.5	1	4.6	1.3	NA	NA
countOE	613	1	584	0.95	39012	64
countOE2	9235	15	"	"	"	"
dffAE	71.2	1	66.1	0.93	472	6.6
dffAE2	143	2.0	"	"	"	"
dffL	0.4	1	2.4	6	NA	NA
dffOE	36.6	1	43.7	1.2	1644	45
dffOE2	208	5.7	"	"	"	"
nandAE	0.6	1	1.0	1.7	12.4	21
nandAE2	1.3	2.2	"	"	"	"
nandL	0.0	-	0.2	-	NA	NA
nandOE	1.0	-	1.7	-	39.0	-

Figure 6.5 : CPU Time Summary

difference between the analysis times of the "cdeAE" and "countAE" examples. This demonstrates that mixed-mode simulation achieves the goal of concentrating the computational effort only where it is desired.

Figure 6.6 shows the CPU time per iteration for the examples in Figure 6.5. The number of iterations is printed in the SPLICE2 program as the number of net "executions" and, in the SPLICE1 program, as the number of "analyses". For the SPICE2 program, the equivalent number of iterations is taken to be the number of Newton iterations for the transient analysis times the number of nodes in the circuit. It can be seen from Figure 6.6 and Figure 6.5 that the CPU time per iteration for the SPLICE2 program and the SPLICE1 program is relatively constant (approximately 1mS to 4mS) for both electrical and logic analyses whereas the ratio of total CPU time between electrical and logic analyses is quite large (approximately 400). This suggests that the primary reason for the difference in simulation speed between logic and electrical analyses is the number of iterations needed to calculate the behavior, not the fundamental model evaluation speed. This is related to the fact that logic waveforms have far fewer points than electrical waveforms for the same simulation and that logic solutions generally do not require iteration to convergence. Figure 6.6 also shows that the SPICE2 program performs substantially more computation for each solution at a net.

The logic simulation speed of the SPLICE2 program is seen in Figure 6.6 to be comparable to that of the SPLICE1 program. The major difference between the SPLICE2 program and the SPLICE1 program is that in the SPLICE2 program, each net is scheduled separately whereas in the SPLICE1 program, the fanout *list* of nets for each element is scheduled. This difference means that nets can be scheduled at more than one time by the scheduling of the fanout lists of the elements that fanin to the net. As a result, the logic net may be processed more times in the SPLICE2 program than in the SPLICE1 program. Most of the speedup of this effect could be achieved in the SPLICE2 program by treating schedules at the current value of simulated time as special cases and not unscheduling the net at its future value of

CPU Time per Iteration (ms)			
Experiment	SPLICE2	SPLICE1	SPICE2
aluRL	1.10	NA	NA
biglogL	1.28	1.32	NA
cdeAE	2.93	1.67	16.3
cdeAE2	3.89	"	"
cdeL	1.25	2.86	NA
cdeLE	3.04	1.70	NA
cdeOE	3.67	1.87	22.1
cdeOE2	4.03	"	"
countAE	2.72	1.66	13.8
countAE2	3.67	"	"
countL	1.00	2.43	NA
countOE	3.48	1.67	20.3
countOE2	3.89	"	"
dffAE	2.49	2.25	13.8
dffAE2	3.41	"	"
dffL	1.03	6.76	NA
dffOE	3.22	2.20	24.0
dffOE2	3.93	"	"

Figure 6.6 : CPU Time Per Iteration

time.

Another way of speeding the logic simulation would be to treat logic nets that have only one fanin from a gate as special cases. Most logic simulators treat nets by default as having only one fanin and when more than one fanin is present, an *implicit-wired-or* element is created to model the multiple fanins. In the mixed-mode environment, it is more convenient to assume that each net (or wire) has more than one fanin possibly with different signal types. The special case is then the situation where there is only one fanin and the extra element that is the net can be collapsed with the logic gate driving it. This optimization should result in about a factor of 2 improvement in the gate level simulation speed since the effective number of gates is cut approximately in half.

Some statistics about memory usage in the SPLICE2 program are shown in Figure 6.7 along with the numbers of elements, nets, and models. Statistics for memory usage and other measurements are summarized in Figure 6.8 and Figure 6.9 for the SPLICE1 program and the SPICE2 program, respectively. The amount of storage required per element is computed by taking the difference between the total memory used for all elements for two large examples and dividing by the difference in the number of elements. For the electrical examples, the SPLICE2 program requires 88 bytes of storage per element, the SPLICE1 program requires 98 bytes of storage per element, and the SPICE2 program requires 108 bytes of storage per element. However, the SPLICE1 program lumps all grounded capacitor elements onto the appropriate net and is able to achieve a much smaller number of elements. The SPLICE2 program and the SPICE2 program could treat grounded capacitors as a special case and achieve both execution speedup and storage savings. For the logic examples, the SPLICE2 program requires 160 bytes of storage per element while the SPLICE1 program requires 110 bytes of storage per element. The difference is primarily attributable to the more compact data structure for the net in the SPLICE1 program. The ratio of the number of nets to elements can be seen from Figure 6.7 to be ≈ 1 for the logic examples.

Experiment	MEM_BLT	MEM_NET	N_ELMS	N_NET_ELMS	N_MODELS
aluRL	14772	39120	125	151	46
biglogL	15028	46264	237	262	41
cdeAE	77752	297928	3300	475	42
cdeAE2	78208	393872	3300	475	41
cdeL	22624	32984	154	180	43
cdeLE	69996	205972	2160	376	61
cdeOE	113348	456408	5617	577	28
cdeOE2	113804	574384	5617	577	27
countAE	47208	114556	1206	193	39
countAE2	47664	149588	1206	193	38
countL	13644	17508	62	88	38
countOE	61852	184796	2215	247	28
countOE2	62308	231492	2215	247	30
dffAE	31672	18436	107	47	33
dffAE2	32128	21932	107	47	32
dffL	10336	8560	11	37	23
dffOE	33848	31380	291	59	28
dffOE2	34304	37468	291	59	27
nandAE	28792	9160	18	29	28
nandAE2	29248	9632	18	29	27
nandL	8536	6844	5	28	19
nandOE	28796	10988	44	31	26

Figure 6.7 : Memory Usage Summary

Experiment	NEXE	N_SCHED	N_UNSCH	N_ELECT	MEM_TOT
biglogL	135592	59211	612	0	36496
cdeAE	1198599	354768	112959	203931	79000
cdeOE	655506	186333	57461	123517	118272
cdeL	3429	1442	64	0	27360
cdeLE	635530	189387	59822	96228	62200
countAE	570577	175263	56132	98209	34776
countOE	349511	104632	33480	65453	52080
countL	1881	875	14	0	16032
dffAE	29419	9858	3216	5166	12272
dffOE	19892	6280	1879	4065	15424
dffL	355	205	2	107	10256
nandAE	371	147	42	94	9624
nandOE	879	295	68	150	10080
nandL	31	18	0	10	8976

Figure 6.8 : SPLICE1 Statistics Summary

Experiment	NUMEL	MFETS	NUMTTP	NUMRTP	NUMNIT	MEM_TOT
cdeAE	3307	712	2322	547	9972	357640
cdeOE	5624	1326	2970	736	10274	611312
countAE	1213	259	1877	414	7970	132056
countOE	2222	517	2539	615	8573	241176
dffAE	115	22	391	59	1362	13492
dffOE	299	66	569	114	1848	32652
nandAE	24	3	74	2	162	3012
nandOE	50	9	86	9	227	5632

Figure 6.9 : SPICE2 Statistics Summary

Performance statistics for the ITA and ITA2 algorithms in the SPLICE2 program are given in Figure 6.10. The data in the column labeled N_ACC_S give the number of solutions accepted at each electrical net for the ITA2 algorithm. These solutions may be recomputed even after being accepted already due to the non-convergence of a fanin net and therefore it is possible for the number of accepted solutions to be higher than the number of solutions actually found. The data in the column labeled N_FL_S give the number of unique solutions actually flushed from the solution buffer. Figure 6.10 shows that the ratio between accepted and flushed solutions for the ITA2 algorithm is less than two, indicating that the speed disadvantage of the ITA2 algorithm compared to the ITA algorithm is not due to unconverged nets re-scheduling their fanouts that have already converged. This is consistent since there is a comparable mechanism in the ITA algorithm and the tolerance criteria are as close as possible. The data in Figure 6.11 in the column labeled TOT_ITER give the total number of iterations for all electrical nets and the data in the column labeled USE/TOT give the fraction of iterations that actually contributed to solutions that were flushed i.e., were useful. Iterations are wasted when the solution on a net is rejected either by excessive truncation error or a fanin that must cut its time-step. The fraction of useful iterations is also not small enough to explain the performance disadvantage of the ITA2 algorithm. The ratio of total iterations for ITA2 and ITA algorithms does mirror the ratio of computation times and the difference is due to the time-step control algorithms. The ITA2 algorithm allows time-steps to be computed based on truncation error whereas the ITA time-step control uses a minimum time-step. Thus, the ITA2 algorithm is potentially more accurate and robust to changes in circuit properties at the expense of computation overhead. The key however, lies in the number of synchronizations (N_SYNC in Figure 6.11) that occur as a result of rejected solutions (N_REJ_S in Figure 6.10). Note that most of the rejected solutions force a time-step which is too small to represent with the current time grid (or local minimum resolvable time) and this causes a synchronization whereby all of the nets are scheduled at the new time point. This is necessary, as has been described in Chapter 4, to guarantee that the ratio of the longest to shortest time-step is bounded by the buffer size and that all solutions can be iterated together. This means that a small fraction of the nets are determining the minimum time-step and the rest of the nets are not fully able, due to the synchronizations, to take advantage of the solution buffer to achieve a longer time-step than the fast changing nets. There is wide latitude in the design of the window time-step control algorithm and it should be possible to reduce the number of synchronizations significantly.

Figure 6.12 shows a pair of histograms for the number of iterations taken by all nets of a D flip-flop (dffAE2) as a function of the time-steps actually taken normalized to the local minimum resolvable time in effect when the solution was obtained and the ideal time-step calculated from the truncation error estimate of the time-step required for the solution at the

Experiment	N_ACC_S	N_REJ_S	N_IT_REJ	N_FL_S
cdeAE	0	0	0	222667
cdeAE2	2700486	2625	406	1430613
cdeLE	0	0	0	95010
cdeOE	0	0	0	128547
cdeOE2	3230480	3513	0	2020790
countAE	0	0	0	103517
countAE2	787447	1593	175	420588
countOE	0	0	0	66212
countOE2	1136873	2548	0	711365
dffAE	0	0	0	4675
dffAE2	18660	114	13	8633
dffOE	0	0	0	3630
dffOE2	23811	215	0	17609
nandAE	0	0	0	56
nandAE2	168	5	0	112
nandOE	0	0	0	112

Figure 6.10 : Electrical Statistics for Solutions

Experiment	N_SYNC	N_WINDOW	TOT_ITER	USE/TOT
cdeAE	0	4000	1075979	0.999
cdeAE2	1684	2171	5113677	0.671
cdeLE	0	4000	568770	0.999
cdeOE	0	4000	320900	0.997
cdeOE2	2121	2602	6606831	0.665
countAE	0	4000	506002	0.999
countAE2	955	1440	1578167	0.738
countOE	0	4000	167378	0.998
countOE2	1781	2236	2365764	0.679
dffAE	0	1000	26435	0.998
dffAE2	71	211	41309	0.862
dffOE	0	1000	9215	0.995
dffOE2	132	275	52135	0.845
nandAE	0	30	193	0.979
nandAE2	2	16	409	0.905
nandOE	0	30	307	0.974

Figure 6.11 : Electrical Statistics for Time-Steps

net also normalized to the local minimum resolvable time. The ideal time-step is an upper bound on the required time-step since breakpoints or other changes at fanin nets can force a smaller time-step to be necessary. It can be seen from Figure 6.12 that most of the time-steps taken are at the value of the local minimum resolvable time while the ideal time-steps are much larger. This indicates that quite a bit of improvement is possible in the window control algorithm.

The OR gate versions of the examples are seen in Figure 6.5 to require relatively more CPU time for ITA2 than the AND gate versions of the examples. This is attributable to the fact that the circuits are more unidirectional and therefore the fraction of nets that are changing is fewer. The internal nets of a NAND gate all need to be iterated together when any input changes, however a NOR gate of arbitrary size has only one internal net to be iterated. The data in the column labeled N_IT_REJ in Figure 6.10 give the number of rejected solu-

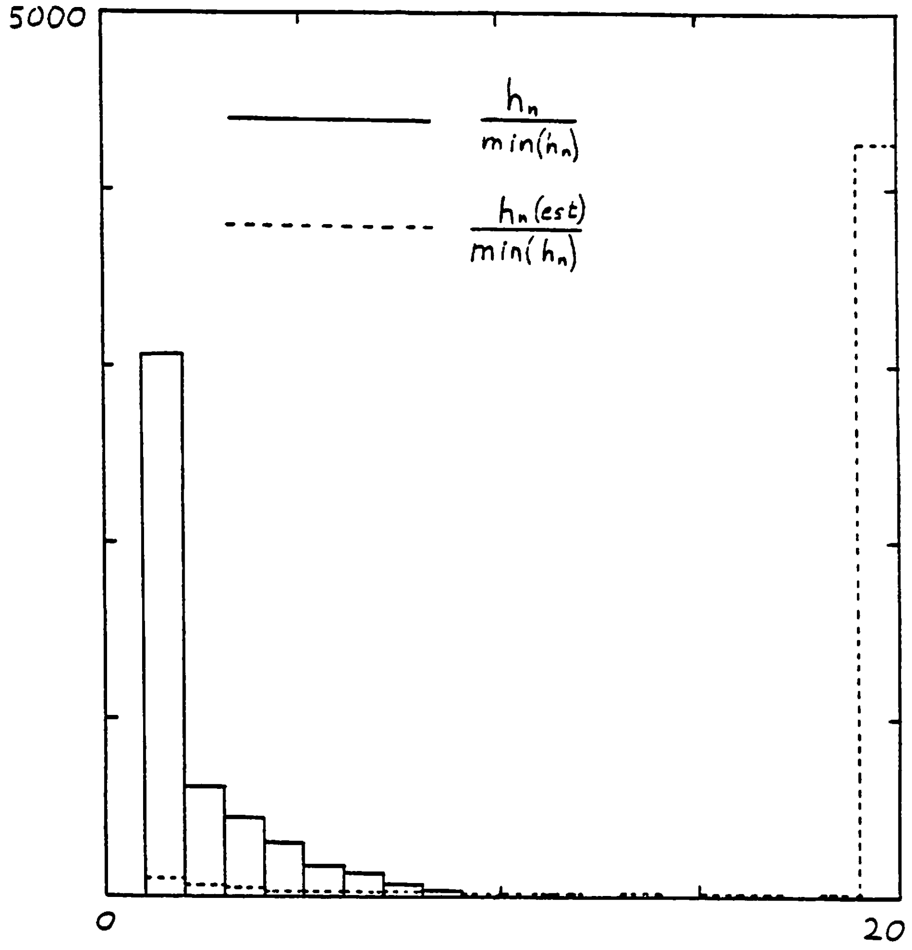


Figure 6.12 : Actual vs. Ideal Time-Step Histogram

Experiment	NEXE	N_TOT_EV	NSCHED	NRESCH
aluRL	83400	250901	124748	41220
biglogL	263447	806249	398898	135211
cdeAE	1084255	13917454	1205344	121075
cdeAE2	5118187	69935592	6651414	1532775
cdeL	5816	19317	8843	2869
cdeLE	580267	7290543	656874	76539
cdeOE	329176	6041251	345949	16765
cdeOE2	6612155	115755096	8192673	1579964
countAE	514278	6555817	570223	55931
countAE2	1581180	21105136	2049977	468627
countL	3496	10046	5173	1611
countOE	175654	3043370	184251	8589
countOE2	2370336	40759740	2948791	578231
dffAE	28566	320399	31646	3071
dffAE2	41777	500411	54331	12529
dffL	388	709	517	114
dffOE	11346	166174	11896	541
dffOE2	52738	893851	63055	10280
nandAE	274	2131	292	10
nandAE2	457	4505	550	84
nandL	45	21	56	3
nandOE	388	4624	414	15

Figure 6.13 : Scheduler Operation Statistics

tions that were rejected due to a maximum iteration count being exceeded. The number of iteration limit rejections is non-zero only for the AND gate versions of the examples. This is consistent with the fact that the strong bilateral coupling present in the AND gate versions leads to off-diagonal terms in the iteration matrices and therefore larger eigenvalues. The iteration matrices for the OR gate versions which are unidirectional are more diagonally dominant and therefore have smaller eigenvalues and do not require as many iterations for solution. It may be possible that, for some circuits, stability requirements rather than truncation error estimates dictate the minimum time-steps required.

Figure 6.13 and Figure 6.14 are summaries of statistics gathered about the operation of the event scheduler in the SPLICE2 program. Some of these results are presented in a different form in Chapter 2 in the description of the design of the event scheduler. The data

Experiment	NSCH_HIT	SCH_HR	NSCH_TT_HIT	SCH_TT_HR	COMP_HR
aluRL	50943	0.585	37671	0.914	0.691
biglogL	180739	0.668	128502	0.95	0.762
cdeAE	147784	0.683	988945	0.999	0.942
cdeAE2	2443830	0.896	3923920	0.998	0.956
cdeL	3741	0.603	2634	0.918	0.702
cdeLE	74606	0.602	532982	0.997	0.923
cdeOE	88884	0.882	245138	0.994	0.961
cdeOE2	3204247	0.96	4853323	0.998	0.983
countAE	69550	0.645	462407	0.998	0.931
countAE2	640437	0.799	1248795	0.996	0.919
countL	1851	0.487	1372	0.852	0.596
countOE	46675	0.804	126218	0.99	0.932
countOE2	1100568	0.93	1765607	0.997	0.97
dffAE	2918	0.381	23980	0.982	0.838
dffAE2	10256	0.512	34282	0.98	0.809
dffL	72	0.154	50	0.439	0.21
dffOE	2455	0.46	6556	0.944	0.733
dffOE2	18559	0.716	37124	0.973	0.869
nandAE	40	0.22	110	0.688	0.439
nandAE2	53	0.172	242	0.752	0.468
nandL	11	0.196	0	0	0.186
nandOE	102	0.464	194	0.907	0.682

Figure 6.14 : Scheduler Performance Statistics

in the column labeled NEXE give the number of times that a net element was processed by the scheduler next event mechanism. This means the number of times that the schedule functions associated with all net models were called. The data in the column labeled N_TOT_EV give the total number of times that net and element evaluation functions were called. The data in the column labeled NSCHED give the number of times that the scheduler was asked to put an event into the schedule queue and the data in the column labeled NRESCH give the number of times that the events were removed from the schedule queue and re-scheduled. The data in the column labeled NSCH_HIT give the number of times that a call to the scheduler occurred and the schedule cache pointer was valid (a hit is when the cache pointer was used as the starting point for the linear list insertion of the event). The data in the column labeled NSCH_HR give the ratio of times that there was a cache hit to the total number of schedules for the schedule cache pointer. The data in the column labeled NSCH_TT_HIT give the number of times that the there was a cache hit for the "This Time" (or zero delay schedule) cache pointer. The data in the column labeled NSCH_TT_HR give the ratio of times that there was a cache hit to the total number of schedules for the "This Time" schedule cache pointer. The data in the column labeled COMP_HR give the composite hit ratio for both of the schedule cache pointers. The hit ratios show that the two element schedule cache works quite adequately for electrical simulation but begins to be inefficient for the larger logic simulation examples. The cached indexed list scheduling method described in Chapter 2 should restore the performance for these cases.

CHAPTER 7

Conclusions

A software program architecture for the simultaneous simulation of circuits at levels ranging from detailed, accurate, electrical simulation to high-level register-transfer models has been described and demonstrated to be effective in the SPLICE2 mixed-mode simulation program. The program architecture accommodates simulation levels that are schematic in nature where the elements to be modeled communicate via signals transmitted along wires. Simulation levels for electrical, logic, and register-transfer levels of simulation have been implemented and demonstrated to work effectively in the SPLICE2 program. Fundamental to these models is the definition of a small number of signal types with the property that each has not only a signal level but also a signal strength. The signal strengths are then used for combining signals at a wire in a uniform way. Other types of simulation can be added easily by defining the signal type, coercions to other levels, and adding the net and element models. One example of another type of simulation that the SPLICE2 program has been used to explore is a discrete electrical simulation method named ELOGIC (for Electrical LOGIC simulation).

Each type of simulation makes different demands on how time is represented and activity in the simulator is scheduled and in a mixed-mode simulator, all of the different types of simulation are carried out simultaneously. In the SPLICE2 program, a representation for time and a scheduling technique is used that is capable of accommodating the different simulation levels. At the electrical level of simulation, the iterated timing analysis algorithm provides a completely decoupled algorithm for the dc-transient simulation problem that is up to two orders of magnitude faster than found with the SPICE2 program. At the logic and register levels of simulation, the performance is around 1000 net solutions per second which, since the current implementation treats all wires as wired-or gates, corresponds to about 2000 gate evaluations per second. This is comparable to the performance of the SPLICE1 program. Thus, the SPLICE2 program has demonstrated that it is possible to perform widely different types of simulation simultaneously in a single program. This is achieved using a program architecture that facilitates the addition of new simulation types without sacrificing either simulation speed or accuracy.

The cached indexed list method for event scheduling appears promising and deserves to be studied more closely. The application of cache-like techniques to algorithms is an important way to improve the efficiency of the algorithm while making its design more simple. This technique should find wider application for general purpose event-driven simulation.

The iterated timing analysis algorithm and other issues for event driven circuit simulation also deser.ve further work. A number of techniques fall under the heading of improvements to the rate of convergence to a solution of the iteration of a system of equations. A faster rate of convergence usually also results in a more accurate result. The coupling method described in Chapter 4 is a new technique that requires further study of its properties and may also have wider application to structural engineering or other problems that use relaxation of systems of equations.

A number of techniques such as successive over-relaxation (sor) and the coupling method described in Chapter 4 may benefit from a variation of the controlling variables of the algorithm as a function of circuit properties. This kind of variation of the algorithm results in non-stationary or semi-iterative methods, such as the Chebyshev semi-iterative method for the solution of linear systems of equations. The idea is that as the solution variable gets closer to the point of contraction, the properties of the iterative process can be measured and the iteration algorithm can be adjusted accordingly.

Over- and under-relaxation can be used to accelerate or stabilize an iterative process, respectively. Further work is required to explore the choice of the largest factor that results in a stable iteration in the context of electrical simulation. Further, estimation of asymptotic values may improve significantly the convergence rate and accuracy for some problems.

The order in which the equations are iterated can have a dramatic effect on the rate of convergence of a system of equations and ways of re-ordering the solution sequence are worth examining. Local properties such as signal level or strength variation may be useful for controlling the equation ordering.

During iteration to convergence, the system of equations being solved behaves like the transient simulation in discrete time, or the *z* domain, of an equivalent circuit. This observation allows one to evaluate changes to algorithms using circuit intuition about stability and slew rate to obtain the fastest convergence rate.

Currently, the solution of equations in the SPLICE2 program is completely decoupled and there are no matrices explicitly represented in the program. It is likely to be advantageous to cluster equations into sub-matrices such as in the original SPLICE1 [1], SAMSON, [13] and RELAX [53] programs. This technique could be used for nets which are so tightly coupled that the iteration sequence is either unstable or converges slowly using the completely decoupled algorithm. How to partition the circuit into these sub-blocks then becomes an important issue as well.

Other areas for further work include extending iterated timing analysis to handle inductors and other "non-nodal" elements such as floating voltage sources. It is clear that relaxation-based circuit analysis techniques can be applied to the problems of AC, noise, and sensitivity calculations. Finally, special purpose hardware for mixed-mode simulation will have a major impact on this field of study.

REFERENCES

1. Newton, A. R., "The Simulation of Large-Scale Integrated Circuits", *ERL Memo No. ERL-M78/52*, July 1978.

2. DeMan, H., G. Arnout and P. Reynaert, "Mixed-Mode Circuit Simulation Techniques and their Implementation in DIANA", *NATO Advanced Study Institute on Computer Design Aids for VLSI Circuits*, Sogesta-Urbino, Italy, July 1980.

3. Saleh, R. A., "Iterated Timing Analysis and SPLICE1", M84/2, Electronics Research Laboratory, UC Berkeley, 1984.

4. Nagel, L. W., "SPICE2: A Computer Program to Simulate Semiconductor Circuits", *ERL Memo UCB/ERL M75/520*(May 1975), University of California, Berkeley.

5. Kleckner, J. E., R. A. Saleh and A. R. Newton, "Electrical Consistency in Schematic Simulation", *Proceedings of ICCC, New York*, September 1982, 30-33.

6. Beyers, J. W. and al., "A 32-bit VLSI CPU Chip", *ISSCC Digest of Technical Papers*, New York, N. Y., February 1981, 104-105.

7. Saleh, R. A., J. E. Kleckner and A. R. Newton, "Iterated Timing Analysis", *Proceedings of ICCAD*, Santa Clara, CA, September 1983, 139.

8. Idleman, T. E., F. S. Jenkins, W. J. McCalla and D. O. Pederson, "SLIC-A Simulator for Linear Integrated Circuits", *IEEE J. Solid-State Circuits SC-6*(Aug. 1971), 188-204.

9. Fan, S. P., M. Y. Hsueh, A. R. Newton and D. O. Pederson, "MOTIS-C: A New Circuit Simulator for MOS LSI Circuits", *Proc. IEEE Int. Symp. Circuits Syst.*, April 1977,

700-703.

10. Boyle, G. R., "Simulation of Integrated Injection Logic", *ERL Memo No. ERL-M78/13*, March 1978.

11. Yang, P., "An Investigation of Ordering, Tearing, and Latency Algorithms for the Time-Domain Simulation of Large Circuits", Tech. Rep. R-891, Univ. of Illinois, Urbana, 1980.

12. Cohen, E., "Performance Limits of a Dedicated CAD System", *Proc. IEEE Int. Symp. Circ. Syst.*, April 1980, 708-711.

13. Sakallah, K. A., "Mixed Simulation of Electronic Integrated Circuits", DRC-02-07-81, Carnegie Mellon University, November, 1981.

14. Vladimirescu, A., "Simulating VLSI Circuits", *PhD Thesis, EECS - UCB*, Berkeley, CA, Jan 1983.

15. Hitchcock, Sr., R. B., "Timing Verification and the Timing Analysis Program", *Proceedings of the 19th Design Automation Conference*, 1982, 594-604.

16. Jouppi, N. P., "Timing Analysis for nMOS VLSI", *Proceedings of the 20th Design Automation Conference*, 1983, 411-418.

17. Szygenda, S. A., "TEGAS2 - Anatomy of a General Purpose Test Generation and Simulation System for Digital Logic", *Proceedings of the Design Automation Conference*, 1972, 116-127.

18. Ratiu, I. M., "VICTOR: Global Redundancy Identification and Test Generation", *ERL Memo No. ERL-M78/52*, May 1983.

19. Nye, W. T., *DELIGHT: An Interactive System for Optimization-Based Engineering Design*, U. C. Berkeley, 1983. PhD Dissertation.

20. Antoniadis, D., S. Hansen and R. Dutton, "Suprem II - A program for IC process modeling and simulation", Tech. Rep. 5019-2, Stanford Electron. Lab., Stanford, California, 1978.

21. Nandgaonkar, S. N., W. G. Oldham and A. R. Neureuther, "Integrated Circuit Process Modeling with SAMPLE", *Proceedings of the 4th Biennial University-Government-Industry Microelectronic Symposium*, Presented at Mississippi State University, May 26, 1981, 32-40.

22. Greenfield, J. A., S. E. Hansen and R. W. Dutton, "Two-Dimensional Analysis for Device Modeling", *Stanford Technical Report G-201-7*(July 1980), 1-184.

23. Chawla, B. R., H. K. Gummel and P. Kozak, "MOTIS - An MOS Timing Simulator", *IEEE Trans. Circuits and Syst. CAS-22*,13 (December 1975), 901-909.

24. Barbacci, M., *The ISPL Language*, Department of Computer Science, Carnegie Mellon University, Pittsburg Pa., 1977.

25. Hill, D. D., *Language and Environment for Multi-Level Simulation*, University Microfilms International, Ann Arbor, Michigan, 1980.

26. Pritsker, A. A. B., *Simulation with GASP IV*, John Wiley, New York, 1976.

27. Birtwistle, G., *SIMULA BEGIN*, Petrocelli/Charter, 641 Lexington Ave., New York, 1973.

28. Arnout, G. and H. DeMan, "The Use of Threshold Functions and Boolean-Controlled Network Elements for Macromodelling of LSI Circuits", *IEEE J. of Solid-State Circuits SC-13*(June 1978), 326-332.

29. Agrawal, V. D., "The Mixed Mode Simulator", *Proc. 17th Design Automation Conf.*, June 1980, 618-625.

30. Daseking, H. W., R. I. Gardner and P. B. Weil, "VISTA: A VLSI CAD System", *IEEE Trans. on CAD CAD-1*,1 (January 1982), 36-52.

31. Engl, W. L., R. Laur and H. K. Dirks, "MEDUSA - A Simulator for Modular Circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems CAD-1*,2 (April 1982), 85-93.

32. Dumlugol, D., H. J. DeMan, P. Stevens and G. G. Schrooten, "Local Relaxation Algorithms for Event-Driven Simulation of MOS Networks Including Assignable Delay Modeling", *IEEE Trans. on CAD CAD-2*,3 (July 1983), 193-202.

33. DeMan, H., L. Darcis, I. Bolsens, P. Reynaert and D. Dumlugol, "A Debugging and Guided Simulation System for MOSVLSI Design", *Proceedings of ICCAD*, Santa Clara, CA, September 1983, 137.

34. Johnson, W. A., J. J. Crowley and J. D. Ray, "Mixed-Level Simulation from a Hierarchical CHDL", *Proceedings of the Fourth International Symposium on Computer Hardware Description Languages*, Palo Alto, California, October 1979.

35. Sasaki, T., A. Yamada, S. Kato, T. Nakazawa, K. Tomita and N. Nomizu, "MIXS: A Mixed Level Simulator for Large Digital System Logic Verification", *Proceedings of the 17th Design Automation Conference*, 1980.

36. Terman, C. J., "RSIM - A Logic-Level Timing Simulator", *Proceedings of the International Conference on Computer Design*, 1983, 437-440.

37. Kleckner, J. E. and K. H. Keller, *Net-List Processor Interface*, U.C. Berkeley, EECS, 1983. Internal Memorandum.

38. Crawford, J., *A Unified Hardware Description Language for CAD Programs*, ERL, U. of California at Berkeley, Aug. 1979.

39. Keller, K., "An Electronic Circuit CAD Framework", PhD Dissertation, UC Berkeley, 1984.

40. Keller, K. H., J. E. Kleckner and T. I. Quarles, *Waveform Analysis Package*, U. C. Berkeley, EECS, 1983. Internal Memo.

41. Foderaro, J. K. and K. L. Sklower, "The Franz Lisp Manual", *4.1BSD Unix User's Manual* 2c(September 1981).

42. Newton, A. R. and D. O. Pederson, "Analysis Time, Accuracy and Memory Requirement Tradeoffs in SPICE2", *Proc. Asilomar Conference*, 1978.

43. Newton, A. R., "Timing, Logic, and Mixed Mode Simulation for Large MOS Integrated Circuits", *NATO Advanced Study Institute on Computer Design Aids for VLSI Circuits*, Sogesta-Urbino, Italy, July 1980.

44. Ulrich, E., "Event Manipulation for Discrete Simulations Requiring Large Numbers of Events", *Comm. of ACM*, Sept. 1978, 777-785.

45. Ulrich, E. G., "Non-Integral Event Timing for Digital Logic Simulation", *Proc. 13th Design Automation Conference*, 1976, 61-67.

46. Szygenda, S. A. and E. W. Thompson, "Digital Logic Simulation in a Time-Based Table-Driven Environment. Part 1. Design Verification", *IEEE Computer*, March 1975, 24-36.

47. Wyman, F. P., *Simulation Modeling: A Guide to Using SIMSCRIPT*, John Wiley, New York, 1970.

48. Schriber, T. J., *A GPSS Primer*, John Wiley, New York, 1976.

49. Vaucher, J. G. and P. Duval, "A Comparison of Simulation Event List Algorithms", *Comm. of ACM 18*,4 (April 1975), 223-230.

50. Wyman, F. P., "Improved Event-Scanning Mechanisms for Discrete Event Simulation", *Comm. of ACM 18*,6 (June 1975), 350-353.

51. Quarles, T., Private Communication, 1983.

52. Sangiovanni-Vincentelli, A. L., "Circuit Simulation", *NATO Advanced Study Institute on Computer Design Aids for VLSI Circuits*, Sogesta-Urbino, Italy, July 1980.

53. Lelarasmee, E. and A. Sangiovanni-Vincentelli, "RELAX: A New Circuit Simulator for Large Scale MOS Integrated Circuits", Memo. ERL-M82/6, UC Berkeley Electronics Research Lab, Berkeley, CA, February 1982.

54. Chua, L. O. and P. M. Lin, *Computer Aided Analysis of Electronic Circuits*, Prentice-Hall, 1975.

55. Ortega, J. M. and W. C. Rheinboldt, *Iterative Solution of Nonlinear Equations in Several Variables*, Academic Press, 1970.

56. Hachtel, G. D. and A. S. Sangiovanni-Vincentelli, "A Survey of Third-Generation Simulation Techniques", *Proceedings IEEE 69*,10 (October 1983).

57. Newton, A. R. and A. L. Sangiovanni-Vincentelli, "Relaxation-Based Electrical Simulation", *IEEE Transactions on Electron Devices ED-30*,9 (Sept. 1983), 1184-1207.

58. Brayton, R. K., F. G. Gustavson and G. D. Hachtel, "A New Efficient Algorithm for Solving Differential-Algebraic Systems Using Implicit Backward Differentiation Formulas", *Proc. IEEE 60*,1 (Jan. 1972), 98-108.

59. Duff, I. S., "A Survey of Sparse Matrix Research", *Proc. of the IEEE*, April 1977.

60. Desoer, C. A. and E. S. Kuh, *Basic Circuit Theory*, McGraw-Hill, 1969.

61. Varga, J., *Matrix Iterative Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey, 1962.

62. Carre, B. A., "The Determination of the Optimum Accelerating Factor for Successive Over-Relaxation", *Computer Journal*, 1961, 73-78.

63. McCalla, W. J., unpublished manuscript.

64. Greenberg, S., Private Communication, November 1983.

65. Weeks, W. T., "Algorithms for ASTAP - A Network Analysis Program", *IEEE Trans. on CT. CT20*(Nov. 1973), 628-634.

66. Breuer, M. A., "General Survey of Design Automation of Digital Computers", *Proc. IEEE 54*,12 (Dec. 1966).

67. Baldwin, R. C., "An Approach to the Simulation of Computer Logic", *AIEE Conference*, 1959.

68. Bose, A. K. and S. A. Szygenda, "Detection of Static and Dynamic Hazards in Logic Nets", *Proceedings of the 14th Design Automation Conference*, 1977, 220-224.

69. Eichelberger, E. B., "Hazard Detection in Combinational and Sequential Switching Circuits", *IBM Journal of Research and Development*, March 1965.

70. *The LOGIS Logic Simulator*, Information Systems Design, Inc., Santa Clara, CA.

71. Staff, *The TTL Data Book for Design Engineers, Second Edition*, Texas Instruments, Dallas, Texas, 1976.

72. Graham, S. L., P. B. Kessler and M. K. McKusick, "gprof: A Call Graph Execution Profiler", *Proceedings of the SIGPLAN 17*,6 (June 1982), 120-126. 82 Symposium on Compiler Construction, SIGPLAN Notices.

APPENDIX A

Splice2 Listing

This appendix contains the C language source code for all of the SPLICE2 program. To obtain this program contact Deborah Dunster at the following address:

EECS Industrial Liason Program
437 Cory Hall
University of California
Berkeley, CA 94720

APPENDIX B

Example Inputs and Outputs

This appendix contains input examples and their outputs from the SPLICE2 and SPLICE1 programs. In addition, selected examples also have hierarchical call graph profiling statistics and some examples have the statistics from the SPICE2 program. To obtain these programs contact Deborah Dunster at the following address:

EECS Industrial Liason Program
437 Cory Hall
University of California
Berkeley, CA 94720

APPENDIX C

Tables of SOR Results

This appendix contains tables of results on the convergence rates for the arrays of resistors described in Chapter 4. In the tables that follow, *sor* refers to the successive-over-relaxation parameter and *couple factor* refers to the heuristic coupling factor both of which are described in more detail in Chapter 4. A negative value for the number of iterations means that convergence was not achieved. The maximum and total numbers of iterations are reported for each combination of sor parameter and coupling factor. The maximum number of iterations refers to largest number of iterations required for the solution of any electrical net. The total number of iterations refers to the sum of the number of iterations required for the solution of all electrical nets.

Maximum DC Iterations										
N	SOR Factor (coupling= 0)									
	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9
1,3	1	5	6	7	9	11	15	21	32	67
1,4	7	5	7	8	10	12	15	22	34	68
1,5	12	10	7	9	11	14	18	23	38	65
1,6	18	15	12	10	11	13	16	25	39	83
1,7	25	21	17	14	12	15	19	27	39	83
1,8	30	26	22	18	15	17	19	23	40	90
1,9	39	31	27	22	19	16	20	27	42	89
1,10	47	39	32	28	23	19	20	26	40	71
1,11	54	44	37	32	27	23	22	27	37	87
1,12	62	48	42	35	31	25	20	23	36	66
1,13	67	59	46	39	34	28	24	25	38	86
1,14	75	62	52	46	38	33	27	24	37	82
1,15	83	68	57	47	40	36	31	27	37	78
1,16	90	73	64	54	48	39	34	27	36	66
1,17	98	82	68	58	48	42	37	31	34	80
1,18	104	88	70	60	50	44	40	33	39	72
1,19	113	92	76	62	54	46	42	36	37	69
1,20	119	99	78	70	56	49	45	39	40	74
2,2	11	9	6	8	10	12	16	21	31	63
2,3	41	34	29	24	19	13	16	22	32	62
2,4	82	71	58	49	41	34	26	21	32	61
2,5	130	107	93	80	67	55	45	34	36	64
2,6	153	134	112	91	78	64	51	35	59	184
2,7	237	204	173	137	116	99	82	67	49	65
2,8	288	245	215	181	144	121	101	83	64	63
2,9	335	286	230	209	168	143	116	97	75	53
2,10	382	338	276	221	189	164	146	114	90	55
3,2	25	21	18	14	9	12	16	19	30	64
3,3	99	83	70	59	50	41	32	22	32	65
3,4	210	175	152	126	105	90	73	57	38	55
3,5	341	287	243	208	175	146	123	98	72	55

Total DC Iterations					
N	SOR Factor (coupling= 0)				
	1.0	1.1	1.2	1.3	1.4
1,3	1	5	6	7	9
1,4	13	10	13	15	19
1,5	33	27	19	24	31
1,6	67	55	44	32	40
1,7	117	94	76	60	51
1,8	173	145	121	97	79
1,9	259	203	176	142	121
1,10	351	287	235	202	168
1,11	454	380	310	264	219
1,12	573	454	396	318	277
1,13	693	601	476	389	345
1,14	851	685	565	517	412
1,15	1002	822	706	561	478
1,16	1188	960	825	701	632
1,17	1377	1144	959	822	667
1,18	1558	1320	1073	910	751
1,19	1797	1463	1212	1006	859
1,20	2000	1652	1356	1172	950
2,2	31	25	18	22	28
2,3	315	256	216	178	143
2,4	1200	1021	830	701	580
2,5	3026	2476	2178	1874	1555
2,6	5233	4568	3784	3099	2607
2,7	11140	9517	7987	6466	5409
2,8	17669	15149	12945	11047	8793
2,9	26189	22464	17915	16406	13180
2,10	37044	32287	26984	21307	18147
3,2	170	138	117	89	60
3,3	2495	2079	1738	1465	1222
3,4	12887	10801	9364	7734	6455
3,5	41338	35065	29695	25284	21279

Total DC Iterations					
N	SOR Factor (coupling= 0)				
	1.5	1.6	1.7	1.8	1.9
1,3	11	15	21	32	67
1,4	23	30	42	66	133
1,5	38	48	65	100	187
1,6	48	62	88	131	271
1,7	63	79	111	164	311
1,8	83	96	123	189	375
1,9	101	123	157	225	428
1,10	139	140	182	261	480
1,11	186	166	208	299	553
1,12	227	183	216	322	560
1,13	281	232	257	347	667
1,14	361	288	276	381	712
1,15	432	364	314	396	756
1,16	502	428	349	433	771
1,17	579	497	408	475	856
1,18	641	568	461	549	906
1,19	719	643	526	580	970
1,20	814	722	612	603	1029
2,2	36	44	61	89	183
2,3	97	107	150	234	470
2,4	468	360	296	442	879
2,5	1252	1024	766	744	1370
2,6	2134	1693	1135	1899	6231
2,7	4611	3783	3069	2210	2475
2,8	7381	6107	5021	3775	3285
2,9	11142	9026	7513	5770	3684
2,10	15736	13922	10743	8409	4738
3,2	75	107	130	202	426
3,3	1013	782	493	750	1414
3,4	5499	4459	3487	2238	3132
3,5	17689	14918	11879	8640	5826

Maximum DC Iterations (sor=1.0)									
N	Coupling Factor								
	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
1,3	1	1	1	1	1	1	1	1	1
1,4	7	7	6	6	5	6	6	5	7
1,5	12	12	11	10	8	7	9	7	11
1,6	18	17	16	14	13	11	10	10	12
1,7	25	23	21	19	17	15	11	12	14
1,8	30	28	26	23	21	18	16	13	12
1,9	39	38	33	28	26	22	19	15	19
1,10	47	42	38	35	32	27	23	20	22
1,11	54	48	45	41	36	30	27	22	23
1,12	62	57	51	46	42	36	29	24	20
1,13	67	66	61	53	47	41	35	27	24
1,14	75	69	65	58	52	46	37	31	28
1,15	83	77	70	66	59	49	41	35	27
1,16	90	83	78	71	64	54	47	39	32
1,17	98	90	83	76	69	59	51	43	35
1,18	104	96	90	82	74	66	55	46	37
1,19	113	104	98	88	81	71	59	48	39
1,20	119	114	102	94	84	78	61	51	42
2,2	11	11	10	9	8	6	7	7	8
2,3	41	37	35	33	31	29	26	23	21
2,4	82	80	73	70	65	63	56	53	47
2,5	130	125	119	112	109	100	93	87	80
2,6	184	175	170	162	151	145	137	128	117
2,7	237	224	222	202	202	191	173	163	158
2,8	288	282	269	251	252	231	225	213	202
2,9	335	325	320	305	293	287	271	256	243
2,10	382	354	354	354	341	329	309	295	285
3,2	25	24	22	21	20	18	17	15	14
3,3	99	96	92	89	83	81	78	73	69
3,4	210	201	197	191	183	178	169	161	155
3,5	341	326	320	310	298	291	281	271	263

Total DC Iterations									
N	Coupling Factor (sor=1.0)								
	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8
1,3	1	1	1	1	1	1	1	1	1
1,4	13	13	11	11	10	11	11	10	13
1,5	33	34	31	27	23	18	26	20	30
1,6	67	63	58	51	47	38	34	35	44
1,7	117	107	95	85	76	66	49	52	59
1,8	173	157	145	132	116	99	86	67	67
1,9	259	243	218	184	169	147	123	97	117
1,10	351	320	288	253	234	201	169	146	151
1,11	454	398	378	346	295	253	224	182	168
1,12	573	523	468	428	383	325	269	219	169
1,13	693	665	609	543	478	409	344	277	229
1,14	851	767	740	651	575	507	413	346	276
1,15	1002	946	861	802	701	579	502	419	323
1,16	1188	1082	1038	927	831	698	603	488	395
1,17	1377	1278	1168	1052	977	815	679	577	459
1,18	1558	1446	1357	1237	1119	959	781	660	521
1,19	1797	1636	1573	1415	1298	1108	891	737	587
1,20	2000	1912	1716	1592	1431	1269	1003	830	656
2,2	31	31	28	25	22	16	21	19	24
2,3	315	288	272	256	237	220	199	179	154
2,4	1200	1152	1065	1008	941	910	813	753	681
2,5	3026	2910	2777	2595	2521	2318	2165	2013	1867
2,6	6231	5991	5773	5542	5105	4922	4623	4313	3950
2,7	11140	10436	10412	9454	9441	8863	8118	7638	7408
2,8	17669	17175	16599	15448	15523	14188	13703	12960	12323
2,9	26189	25520	25106	23839	22880	22132	21287	20021	18877
2,10	37044	34330	33795	34239	32341	31356	29284	28574	27274
3,2	170	162	154	138	131	126	110	102	89
3,3	2495	2427	2321	2230	2093	2055	1978	1816	1732
3,4	12887	12401	12158	11805	11303	10880	10430	9859	9600
3,5	41338	39710	39067	37808	36430	35488	34388	33135	32126

Maximum DC Iterations										
N	Coupling factor ("exact coupling" SOR=1.0)									
	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
1,3	1	1	1	1	1	1	1	1	1	1
1,4	7	7	6	6	6	7	7	8	8	8
1,5	12	11	10	8	8	12	11	13	18	19
1,6	18	16	14	12	9	11	14	19	29	90
1,7	25	22	19	16	11	13	20	36	-1	-1
1,8	30	27	23	20	16	14	22	82	-1	-1
1,9	39	34	28	24	19	17	26	-1	-1	-1
1,10	47	41	35	28	23	17	34	-1	-1	-1
1,11	54	48	42	33	27	18	36	-1	-1	-1
1,12	62	52	46	39	29	22	40	-1	-1	-1
1,13	67	61	52	42	33	25	43	-1	-1	-1
1,14	75	68	61	49	37	26	49	-1	-1	-1
1,15	83	74	65	54	42	29	47	-1	-1	-1
1,16	90	82	71	62	45	33	67	-1	-1	-1
1,17	98	89	78	65	49	36	68	-1	-1	-1
1,18	104	96	83	71	54	39	42	-1	-1	-1
1,19	113	99	88	76	55	43	57	-1	-1	-1
1,20	119	109	93	79	67	46	64	-1	-1	-1
2,2	11	10	9	7	7	8	8	14	19	25
2,3	41	38	34	30	27	23	18	11	20	29
2,4	82	78	71	66	60	53	45	38	28	26
2,5	130	124	116	108	99	88	80	68	56	43
2,6	184	175	164	154	143	128	118	102	87	71
2,7	237	228	208	197	184	168	156	140	124	102
2,8	288	280	254	250	235	218	199	180	156	133
2,9	335	326	308	293	270	262	244	221	191	163
2,10	382	387	360	343	326	301	285	256	229	198
3,2	25	23	22	20	18	16	13	10	11	13
3,3	99	94	90	87	82	75	69	64	57	52
3,4	210	200	193	182	178	168	157	146	139	127
3,5	341	324	312	307	292	279	265	256	239	226

Total DC Iterations									
N	Coupling factor ("exact coupling" SOR=1.0)								
	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9
1,3	1	1	1	1	1	1	1	1	1
1,4	13	11	11	11	13	13	15	15	16
1,5	30	27	21	21	31	28	36	49	53
1,6	59	50	42	33	41	51	73	108	349
1,7	100	86	71	48	57	89	169	-1	-1
1,8	151	132	109	86	69	119	461	-1	-1
1,9	225	184	157	123	106	161	-1	-1	-1
1,10	306	262	210	168	122	244	-1	-1	-1
1,11	402	352	280	224	152	301	-1	-1	-1
1,12	481	435	353	269	198	341	-1	-1	-1
1,13	632	530	428	339	248	430	-1	-1	-1
1,14	773	664	522	414	284	536	-1	-1	-1
1,15	898	795	647	496	341	459	-1	-1	-1
1,16	1085	936	786	569	405	627	-1	-1	-1
1,17	1237	1102	900	676	481	719	-1	-1	-1
1,18	1449	1255	1074	778	554	609	-1	-1	-1
1,19	1573	1419	1212	866	632	848	-1	-1	-1
1,20	1824	1572	1320	1058	721	940	-1	-1	-1
2,2	28	25	19	21	24	24	36	49	65
2,3	291	261	232	205	170	137	81	145	223
2,4	1126	1023	953	877	753	652	532	407	327
2,5	2877	2702	2499	2293	2044	1835	1583	1288	977
2,6	5974	5562	5251	4850	4344	4031	3430	2932	2405
2,7	10666	9693	9286	8552	7843	7277	6531	5652	4578
2,8	17215	15717	15300	14437	13342	12190	10927	9479	7928
2,9	25377	24205	22945	21175	20411	18965	17194	14812	12644
2,10	36578	34637	32906	31270	28571	27211	24555	21905	18789
3,2	152	148	131	117	103	88	64	70	86
3,3	2360	2270	2193	2061	1869	1733	1605	1433	1273
3,4	12412	11884	11301	10952	10240	9681	9043	8514	7864
3,5	39607	38311	37492	35630	34089	32409	31211	29089	27533

Maximum DC Iterations									
N	SOR Factor (coupling= 0.3)								
	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8
1,3	1	5	6	7	9	11	15	21	32
1,4	6	7	8	10	13	17	25	40	115
1,5	10	8	11	12	16	22	33	70	-1
1,6	14	10	10	11	17	25	40	119	-1
1,7	19	15	11	13	18	28	46	163	-1
1,8	23	19	15	15	21	28	46	233	-1
1,9	28	23	19	17	21	25	43	292	-1
1,10	35	27	23	18	20	30	49	367	-1
1,11	41	32	27	22	25	29	52	482	-1
1,12	46	36	30	24	22	30	52	-1	-1
1,13	53	41	33	27	21	28	50	-1	-1
1,14	58	47	37	31	24	26	50	-1	-1
1,15	66	50	41	34	27	28	53	-1	-1
1,16	71	54	46	38	31	36	50	-1	-1
1,17	76	66	51	41	34	30	46	-1	-1
1,18	82	69	55	43	37	31	53	-1	-1
1,19	88	72	59	46	39	32	49	-1	-1
1,20	94	73	62	49	41	35	52	-1	-1
2,2	9	7	9	10	14	21	31	66	-1
2,3	33	28	22	16	15	18	29	60	-1
2,4	70	57	47	38	30	21	29	53	-1
2,5	112	93	78	63	50	39	28	50	-1
2,6	162	138	112	88	74	58	43	52	239
2,7	202	170	140	122	95	77	58	47	219
2,8	251	214	182	145	115	96	74	46	209
2,9	305	264	216	177	138	115	87	60	154
2,10	354	298	234	208	172	129	104	73	205
3,2	21	17	13	11	13	16	26	54	253
3,3	89	72	60	50	40	30	24	37	101
3,4	191	159	130	110	90	72	55	36	81
3,5	310	264	217	191	147	123	97	68	75

Total DC Iterations									
N	SOR Factor (coupling= 0.3)								
	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8
1,3	1	5	6	7	9	11	15	21	32
1,4	11	13	15	19	24	32	48	78	221
1,5	27	21	29	33	44	61	89	200	-1
1,6	51	37	36	42	61	89	144	430	-1
1,7	85	65	46	59	80	118	206	771	-1
1,8	132	105	80	77	102	135	239	1303	-1
1,9	184	149	121	100	126	156	279	1930	-1
1,10	253	205	168	129	147	207	354	2784	-1
1,11	346	266	225	176	189	238	403	4106	-1
1,12	428	334	272	218	192	245	443	-1	-1
1,13	543	430	339	268	216	277	500	-1	-1
1,14	651	536	405	332	260	297	540	-1	-1
1,15	802	589	490	404	313	339	589	-1	-1
1,16	927	691	604	479	380	429	593	-1	-1
1,17	1052	881	716	555	446	407	647	-1	-1
1,18	1237	1019	799	629	513	451	727	-1	-1
1,19	1415	1125	912	718	583	476	740	-1	-1
1,20	1592	1236	1029	802	656	537	836	-1	-1
2,2	25	21	25	30	40	59	89	194	-1
2,3	256	208	165	121	111	132	218	447	-1
2,4	1008	821	667	543	420	271	378	691	-1
2,5	2595	2147	1811	1443	1150	892	622	1108	-1
2,6	5542	4683	3854	3001	2508	1941	1403	1478	7370
2,7	9454	7991	6624	5714	4398	3525	2636	1840	8259
2,8	15448	13086	11197	8892	6999	5862	4443	2704	10122
2,9	23839	20459	16760	13869	10790	8940	6729	4549	10783
2,10	34239	28864	22447	20059	16464	12265	9769	6803	13076
3,2	138	110	88	74	90	109	174	315	1728
3,3	2230	1824	1485	1222	987	741	563	850	2345
3,4	11805	9840	8016	6735	5499	4426	3323	1945	4413
3,5	37808	32300	26568	23300	17925	14918	11743	8151	7309

Maximum DC Iterations										
N	SOR Factor (coupling= 0.5)									
	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9
1,3	1	5	6	7	9	11	15	21	32	67
1,4	6	7	9	12	16	22	34	81	-1	-1
1,5	7	9	11	16	24	37	97	-1	-1	-1
1,6	11	10	14	17	30	58	-1	-1	-1	-1
1,7	15	12	15	22	33	82	-1	-1	-1	-1
1,8	18	13	16	22	34	104	-1	-1	-1	-1
1,9	22	17	17	23	37	145	-1	-1	-1	-1
1,10	27	21	18	24	39	180	-1	-1	-1	-1
1,11	30	24	19	27	43	204	-1	-1	-1	-1
1,12	36	27	21	24	42	257	-1	-1	-1	-1
1,13	41	32	24	28	45	312	-1	-1	-1	-1
1,14	46	34	27	28	47	371	-1	-1	-1	-1
1,15	49	38	30	27	44	-1	-1	-1	-1	-1
1,16	54	42	34	29	38	-1	-1	-1	-1	-1
1,17	59	48	38	30	45	-1	-1	-1	-1	-1
1,18	66	50	41	32	49	-1	-1	-1	-1	-1
1,19	71	53	43	33	38	-1	-1	-1	-1	-1
1,20	78	56	45	35	40	-1	-1	-1	-1	-1
2,2	6	9	10	14	25	34	94	-1	-1	-1
2,3	29	23	17	15	20	38	87	-1	-1	-1
2,4	63	50	39	30	20	31	65	-1	-1	-1
2,5	100	81	65	51	39	32	61	-1	-1	-1
2,6	145	120	95	75	60	42	57	-1	-1	-1
2,7	191	155	128	99	79	58	52	-1	-1	-1
2,8	231	195	159	129	98	75	55	-1	-1	-1
2,9	287	232	193	162	119	91	59	301	-1	-1
2,10	329	276	222	178	151	107	74	-1	-1	-1
3,2	18	14	9	13	16	26	52	-1	-1	-1
3,3	81	68	53	43	33	21	33	85	-1	-1
3,4	178	147	119	98	78	59	38	66	-1	-1
3,5	291	240	204	168	132	105	76	59	-1	-1

Total DC Iterations										
N	SOR Factor (coupling= 0.5)									
	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9
1,3	1	5	6	7	9	11	15	21	32	67
1,4	11	13	17	23	30	42	66	157	-1	-1
1,5	18	24	30	44	64	101	283	-1	-1	-1
1,6	38	35	50	64	107	216	-1	-1	-1	-1
1,7	66	54	65	92	153	375	-1	-1	-1	-1
1,8	99	72	80	111	177	582	-1	-1	-1	-1
1,9	147	111	107	143	241	948	-1	-1	-1	-1
1,10	201	154	126	180	290	1371	-1	-1	-1	-1
1,11	253	199	153	201	352	1781	-1	-1	-1	-1
1,12	325	249	184	209	379	2470	-1	-1	-1	-1
1,13	409	324	237	272	402	3279	-1	-1	-1	-1
1,14	507	379	295	308	470	4246	-1	-1	-1	-1
1,15	579	457	351	316	491	-1	-1	-1	-1	-1
1,16	698	550	428	339	462	-1	-1	-1	-1	-1
1,17	815	665	517	387	571	-1	-1	-1	-1	-1
1,18	959	732	582	440	615	-1	-1	-1	-1	-1
1,19	1108	816	660	484	622	-1	-1	-1	-1	-1
1,20	1269	922	735	551	671	-1	-1	-1	-1	-1
2,2	16	25	30	40	67	98	279	-1	-1	-1
2,3	220	173	122	111	136	248	660	-1	-1	-1
2,4	910	725	555	420	269	424	894	-1	-1	-1
2,5	2318	1877	1498	1184	891	669	1340	-1	-1	-1
2,6	4922	4068	3245	2533	2012	1395	1693	-1	-1	-1
2,7	8863	7212	5949	4614	3651	2633	2024	-1	-1	-1
2,8	14188	11953	9571	7812	5990	4480	2634	-1	-1	-1
2,9	22132	18139	15009	12373	9258	6995	4452	21632	-1	-1
2,10	31356	26453	20665	16960	14223	10123	6790	-1	-1	-1
3,2	126	95	57	88	109	176	343	-1	-1	-1
3,3	2055	1691	1345	1065	803	484	774	2096	-1	-1
3,4	10880	9115	7379	6054	4750	3619	2265	3655	-1	-1
3,5	35488	29344	24950	20464	16025	12746	9188	5804	-1	-1

Maximum DC Iterations										
N	SOR Factor (coupling= 0.7)									
	1.0	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9
1,3	1	5	6	7	9	11	15	21	32	67
1,4	5	8	9	12	19	28	59	-1	-1	-1
1,5	7	11	15	21	39	136	-1	-1	-1	-1
1,6	10	14	19	32	93	-1	-1	-1	-1	-1
1,7	12	16	25	46	-1	-1	-1	-1	-1	-1
1,8	13	17	27	60	-1	-1	-1	-1	-1	-1
1,9	15	18	28	75	-1	-1	-1	-1	-1	-1
1,10	20	24	32	101	-1	-1	-1	-1	-1	-1
1,11	22	22	31	152	-1	-1	-1	-1	-1	-1
1,12	24	28	33	141	-1	-1	-1	-1	-1	-1
1,13	27	22	32	174	-1	-1	-1	-1	-1	-1
1,14	31	23	32	-1	-1	-1	-1	-1	-1	-1
1,15	35	25	39	227	-1	-1	-1	-1	-1	-1
1,16	39	30	34	249	-1	-1	-1	-1	-1	-1
1,17	43	32	45	-1	-1	-1	-1	-1	-1	-1
1,18	46	35	45	353	-1	-1	-1	-1	-1	-1
1,19	48	38	38	-1	-1	-1	-1	-1	-1	-1
1,20	51	39	48	464	-1	-1	-1	-1	-1	-1
2,2	7	10	14	27	36	-1	-1	-1	-1	-1
2,3	23	17	15	21	43	172	-1	-1	-1	-1
2,4	53	41	30	19	33	84	-1	-1	-1	-1
2,5	87	70	55	39	33	79	-1	-1	-1	-1
2,6	128	103	81	61	42	66	-1	-1	-1	-1
2,7	163	133	106	80	58	69	-1	-1	-1	-1
2,8	213	172	136	104	80	55	-1	-1	-1	-1
2,9	256	202	169	129	92	58	-1	-1	-1	-1
2,10	295	244	202	141	109	72	-1	-1	-1	-1
3,2	15	10	13	16	26	57	-1	-1	-1	-1
3,3	73	57	46	35	22	32	87	-1	-1	-1
3,4	161	133	105	86	64	44	55	-1	-1	-1
3,5	271	228	186	150	114	85	54	-1	-1	-1

Total DC Iterations									
N	SOR Factor (coupling= 0.7)								
	1.1	1.2	1.3	1.4	1.5	1.6	1.7	1.8	1.9
1,3	1	6	7	9	11	15	21	32	67
1,4	15	18	24	36	54	113	-1	-1	-1
1,5	30	42	57	107	392	-1	-1	-1	-1
1,6	50	70	122	358	-1	-1	-1	-1	-1
1,7	74	113	209	-1	-1	-1	-1	-1	-1
1,8	84	144	334	-1	-1	-1	-1	-1	-1
1,9	116	186	488	-1	-1	-1	-1	-1	-1
1,10	173	230	746	-1	-1	-1	-1	-1	-1
1,11	179	264	1150	-1	-1	-1	-1	-1	-1
1,12	229	282	1360	-1	-1	-1	-1	-1	-1
1,13	210	328	1842	-1	-1	-1	-1	-1	-1
1,14	242	370	-1	-1	-1	-1	-1	-1	-1
1,15	287	447	2845	-1	-1	-1	-1	-1	-1
1,16	363	407	3328	-1	-1	-1	-1	-1	-1
1,17	425	525	-1	-1	-1	-1	-1	-1	-1
1,18	487	601	4485	-1	-1	-1	-1	-1	-1
1,19	566	613	-1	-1	-1	-1	-1	-1	-1
1,20	622	705	6712	-1	-1	-1	-1	-1	-1
2,2	30	40	71	104	-1	-1	-1	-1	-1
2,3	122	106	160	288	1327	-1	-1	-1	-1
2,4	589	428	263	432	1170	-1	-1	-1	-1
2,5	1634	1257	896	723	1668	-1	-1	-1	-1
2,6	3397	2741	2045	1373	1935	-1	-1	-1	-1
2,7	6207	4981	3691	2658	2255	-1	-1	-1	-1
2,8	10439	8189	6263	4783	2751	-1	-1	-1	-1
2,9	15794	13108	9796	7072	4353	-1	-1	-1	-1
2,10	23488	19263	13539	10370	6633	-1	-1	-1	-1
3,2	67	73	109	175	366	-1	-1	-1	-1
3,3	1436	1143	865	528	749	1928	-1	-1	-1
3,4	8189	6475	5272	3936	2624	3126	-1	-1	-1
3,5	27772	22804	18215	13766	10210	5445	-1	-1	-1

APPENDIX D

Adding Models to SPLICE2

The purpose of this appendix is to show how a model is added to the SPLICE2 program. The procedure will be explained by showing how the electrical diode model would be added if it were not already one of the elements in the model library. The diode model is a good example for demonstration since it is small enough to fit conveniently in this appendix yet complicated enough to have examples of all of the features used in modeling devices.

The usual way to create a new model is to copy an existing model with the proper number of terminals and then edit the contents of that model to become the new model. This description will assume that the splice2 files are in a directory named "splice2". The first step is to change your working directory into the directory that contains the models for the SPLICE2 program. This is done by typing "cd splice2/src/dev". This directory is the easiest one in which to create new device models. The goal will be to produce a compiled object for the model that can then be used with the simulator.

Model files are created with file names that are descriptive of the models in them and with a suffix ".mod" to indicate that the file is a model file. The file is made up of three parts. The first part is the include directives and external variable declarations required for the proper compilation of the C program text of the model. The second part is a data structure declaration for the element and model data structures. Information is included with the data structure declarations that is used to build a table describing this device to the SPLICE2 program routines that allocate and initialize the elements and models that are read in from the user. The third part is the function that is used by the net element models to evaluate the signal contribution of this element to the net and an optional initialization function that is applied to all models and elements created. These parts are described below for the diode model.

The electrical diode model is a two-terminal device that has been copied from the SPICE2 program diode model and is already available in this directory. The C language source code for this model follows:

```
/*
 * diode.mod - Diode models
 * Copyright (C) 1983 James Kleckner
 * All rights reserved.
 */
#ifndef lint
static    char     SccsId[] =
"SccsId @(#)diode.mod        2.1 (Splice2/Berkeley) 1/11/84";
#endif    not lint

#include <math.h>
#include "spl2types.h"
#include "splice.h"
#include "elminfo.h"
#include "signals.h"
#include "modinfo.h"
#include "stinfo.h"
#include "scheds.h"
#include "spltime.h"
#include "elect.h"

double                      ElcPnJuncLimit1();
extern    Boolean           SetNoGo;

/* BEGIN_HEADER_DECLARATION */
struct    ElmElcDiode       {                     /* ATTRIBUTES = ELMOBJECT */
          struct    ElmBasic bas;
          float                     lastV;
/* PARAMETERS */
          float                     area;
}         ;
typedef   struct    ElmElcDiode     ElmElcDiode;
struct    ModElcDiode       {       /* ATTRIBUTES = MODOBJECT */
                                    /* INITFUNC */
                                    /* NETS  from    CIRCUIT INANDOUT
                                                 to      CIRCUIT INANDOUT  */
          struct    ModBasic bas;
          float                     vtherm;
          float                     vcrit;
          float                     f1;
          float                     f2;
          float                     f3;
/* PARAMETERS */
          float                     area;           /* RANGE (0,>) */
```

```
            float                   is;             /* RANGE (0,>) */
            float                   n;              /* RANGE (0,10) */
            float                   tt;             /* RANGE [0,>) */
            float                   cjo;            /* RANGE [0,>) */
            float                   vj;             /* RANGE (0,>) */
            float                   m;              /* RANGE (0,.9] */
            float                   eg;             /* RANGE [.1,5] */
            float                   xti;            /* RANGE (0,10] */
            float                   fc;             /* RANGE [0,.95] */
            float                   bv;             /* RANGE [0,>] */
            float                   ibv;            /* RANGE [0,>] */
            float                   couplefactor;   /* RANGE [0,100]*/
            float                   uselimiting;
/* Not implemented: */
/*          float                   rs;             /* RANGE (0,>] */
/*          float                   kf;             /* RANGE (0,>] */
/*          float                   af;             /* RANGE (0,>] */
}           ;
typedef  struct   ModElcDiode       ModElcDiode;
/* END_HEADER_DECLARATION */

/*
 * ElmElcDiode - Diode model.
 */
ELM_EVAL_DECLARE(ElcDiode)
{
    union   SigStruct               sig0,    sig1;
    struct  ModElcDiode             *diode;
    int                             ioff;
    double                          area,    arg,     bv,   capd,  cd;
    double                          csat,    czero;
    double                          czof2,   evd,     evrev;
    double                          f2,      f3,      fcpb, gd;
    double                          pb,      sarg,    tau;
    double                          vcrit,   vd,      vdtemp,      vte;
    double                          xm,      vt,      cdeq;

#ifdef   DEBUG
    IFDEBUG( ELECTDEBUG ) {
            STRUCTASSERT(elm,ElmElcDiode);
    }
#endif   DEBUG
    diode = (struct  ModElcDiode *) elm->bas.mod;
    /*
     * dc model parameters
     */
    ioff  = 0;
    area  = elm->area;
    csat  = diode->is * area;
    vt    = diode->vtherm;
    vte   = diode->n  * vt;
    bv    = diode->bv;
    vcrit = diode->vcrit;
```

```
/*
 * Compute branch voltage.
 */
if ( ioff != 0 ) {
        vd = 0.0;
}
else {
        NET_EVAL_FUNC(elm->bas.fan.in.lis[0],SIG_NET_ELECTRIC,&sig0,SplCurTime);
        NET_EVAL_FUNC(elm->bas.fan.in.lis[1],SIG_NET_ELECTRIC,&sig1,SplCurTime);
        vd = sig0.netElc.v - sig1.netElc.v;
        /*
         * Compute new nonlinear branch voltage. (bypass not implemented)
         *  If limiting, use it.
         */
        if ( diode->uselimiting != 0 ) {
            if ( bv == 0.0 || vd >= MIN(0.0, -bv + 10.0 * vte) ) {
                vdtemp = ElcPnJuncLimit1( vd, elm->lastV, vte, vcrit );
            }
            else {
                vdtemp = -( vd + bv );
                vdtemp =ElcPnJuncLimit1(vdtemp, -(elm->lastV + bv), vte, vcrit);
                vdtemp = -( vdtemp + bv );
            }
            if ( vdtemp != vd ) {
                ElcNoConvergence = 1;
                vd = vdtemp;
            }
        }
}
/*
 * Compute dc current and derivatives
 */
if ( vd >= -5.0 * vte ) {
        evd = exp( vd / vte );
        cd  = csat * ( evd - 1.0 );
        gd  = csat * evd / vte;
}
else {
        if ( bv == 0.0 || vd >= -bv) {
            gd = -csat / vd;
            cd = gd * vd;
        }
        else {
            evrev = exp( -(bv + vd) / vt );
            cd    = -csat * ( evrev - 1.0 + bv/vt );
            gd    = csat * evrev / vt;
        }
}
/*
 * Charge storage elements
 */
tau   = diode->tt;
czero = diode->cjo * area;
```

```
	pb	= diode->vj;
	xm	= diode->m;
	fcpb	= diode->fc;
	if ( vd < fcpb ) {
		arg  = 1.0 - vd / pb;
		sarg = exp( -xm * log(arg) );
		capd = tau * gd + czero * sarg;
	}
	else {
		f2 = diode->f2;
		f3 = diode->f3;
		czof2 = czero/f2;
		capd = tau * gd + czof2 * (f3 + xm * vd/pb);
	}
	elm->lastV = vd;
	/*
	 * Store result.
	 */
	cdeq = cd - gd * vd;
	if ( net == elm->bas.fan.in.lis[0] ) {
		cd = cdeq - gd * sig1.netElc.v - capd * sig1.netElc.vdot;
	}
	else {
		DASSERT( net == elm->bas.fan.in.lis[1] );
		cd = -cdeq - gd * sig0.netElc.v - capd * sig0.netElc.vdot;
	}
	result->elmElc.g   = gd;
	result->elmElc.i   = cd;
	result->elmElc.cap = capd;
	return( SIG_ELM_ELECTRIC );
}
/*
 * Initialize the Diode model by precalculating the thermal voltage.
 */
MOD_INIT_DECLARE(ElcDiode,elm)
{
	struct  ModElcDiode		*diode;
	int				i;
	double				pb,	xm,	fc,	xfc;
	double				bv,	vte,	vt,	csat;
	double				cbv,	xbv,	xcbv, tol;

	if ( StType(elm) == StNamTyp("ModElcDiode") ) {
		diode = (struct ModElcDiode *) elm;
		/*
		 * Precompute the thermal voltage as k*T/q
		 */
		vt		= 1.38e-23 * 300 / 1.602e-19;
		diode->vtherm = vt;
		pb		= diode->vj;
		xm		= diode->m;
		fc		= diode->fc;
		diode->fc	= fc * pb;
```

```
        xfc             = log(1.0 - fc);
        diode->f1       = pb * (1.0 - exp((1.0 - xm) * xfc))/(1.0 - xm);
        diode->f2       = exp((1.0+xm) * xfc);
        diode->f3       = 1.0 - fc * (1.0 + xm);
        csat            = diode->is;
        vte             = diode->n * vt;
        diode->vcrit    = vte * log( vte/(sqrt(2.0) * csat) );
        bv              = diode->bv;
        if ( bv == 0.0 ) {
            return;
        }
        cbv = diode->ibv;
        if ( cbv  <csat * bv / vt ) {
            cbv = csat * bv / vt;
            Error( NONFATAL, "EliElcDiode: %s%g%s",
                        "warning - ibv increased to ", cbv,
                        "to resolve incompatibility with specified isat" );
            xbv = bv;
        }
        else {
            tol   = 0.0001 * cbv;
            xbv  = bv - vt * log( 1.0 + cbv / csat );
            for ( i = 0; i <25; i++ ) {
                    xbv  = bv - vt * log( cbv/csat + 1.0 - xbv/vt );
                    xcbv = csat * ( exp(( bv - xbv )/vt) - 1.0 + xbv/vt );
                    if ( ABS(xcbv - cbv) <= tol ) {
                        break;
                    }
            }
            if ( i >= 25 ) {
                    SetNoGo = TRUE;
                    Error( NONFATAL, "EliElcDiode: warning: %s\ nbv=%g ibv=%g",
                        "unable to match forward and reverse diode regions",
                        xbv, xcbv );
            }
        }
        diode->bv = xbv;
    }
}

/*
 * ElcPnJuncLimit1 - this routine limits the change-per-iteration
 * of device pn-junction voltages.
 * New method.
 */
double
ElcPnJuncLimit1( vnew, vold, vt, vcrit )
    double          vnew,   vold,   vt,     vcrit;
{
    double          delv,   vlim,   arg;

    if ( vnew  > vcrit ) {
            vlim = vt + vt;
```

```
257             delv = vnew - vold;
258             if ( ABS(delv) > vlim ) {
259                 if ( vold <= 0.0 ) {
260                         vnew = vt * log( vnew/vt );
261                 }
262                 else if ( vold < vcrit ) {
263                         vnew = vcrit;
264                 }
265                 else if ( (arg = 1.0 + delv/vt) > 0.0 ) {
266                         vnew = vold + vt * log( arg );
267                 }
268                 else {
269                         vnew = vcrit;
270                 }
271             }
272     }
273     return( vnew );
274 }
```

The first part of the model on lines 11-20 is a set of include directives that gather all of the external type declarations necessary for the compilation of the model by the C compiler. The file "math.h" is included since exponential and logarithm functions are used for this model. The files "spl2types.h", "splice.h", "elminfo.h", "signals.h", "modinfo.h", "stinfo.h", "scheds.h", and "spltime.h" are required for definition of data structures, constants, and macro definitions that are used when writing models. The file "elect.h" is included to access the electrical analysis variable, "ElcNoConvergence", that tells the electrical net that convergence has not been achieved. This occurs when the limiting algorithm for the diode nonlinearity is used to improve the convergence rate. The declaration for the function "ElcPnJuncLimit1" on line 22 allows the diode routine to use the limiting function that is declared further down in the file. The declaration on line 23 of the "SetNoGo" variable is to allow the diode initialization routine to flag the setup pass of the simulator that there has been an error reading in a diode element or model and that execution should not be begun.

The data structure declarations for the electrical diode appear on lines 25-64. These declarations are read by the "stmake" program to build a table describing this model to the read routines. The characters on line 25 form a pattern that is used to signal the "stmake" program that information should begin to be collected and the pattern on line 64 indicates that information should cease being gathered. These patterns must be given exactly as shown to work properly. The element data structure declaration is begun on line 26 and the model data structure declaration is begun on line 33. The element data structure name must begin with the prefix "Elm" and the model data structure name must begin with the prefix "Mod". The rest of the name of the data structure is the name that the user requests and is used by the read routines. Thus the name of this model is "ElcDiode". There are some naming conventions for model names to help control the complexity of managing the different models. By convention, electrical model names begin with the prefix "Elc", logic model names begin with the prefix "Log", and RTL model names begin with the prefix "RTL". Further, if an element model is a net element, then the characters "Net" precede the prefix.

The information for the device table is passed in the comments of the data structure declaration so that the same text can be used as input to the compiler. The general form of the information is a keyword followed by an optional equal sign followed by the text associated with the keyword up to the closing comment characters. The keyword ATTRIBUTES must be present for both the element and model data structures and may take on the values ELMOBJECT, MODOBJECT, NETELEMENT, or NETMODEL to indicate that the data structure types is an element, model, net element, or net model, respectively. Every element and model data structure must include a reference to the proper basic structure as the first member of the declaration which must be named "bas". This is done on lines 27 and 37 for the diode element and model. The basic structure types are ElmBasic, ModBasic, ElmNetBasic, and ModNetBasic for element, model, net element, and net model basic structures, respectively. The inclusion of the basic structure guarantees that all of the information that must be common to all data structures of that type is included automatically. This information includes such things as fanin and fanout lists and hash table links.

Static data storage that must be allocated for elements or models follows the basic structure member. For the diode example, line 28 declares that a floating point variable called "lastV" is available for every diode and lines 38-42 allocate storage for every diode model that is created. The storage for parameters is allocated after the keyword PARAMETERS. Each parameter is declared by giving the C language type, the name of the member, and an optional range string. The member name is picked directly out of the declaration by the "stmake" program and used as the name for the parameter during the input and setup process. This makes it difficult to make an error in connecting the data supplied by the user with the data actually used in the model routine. The convention for element parameters is that all parameters on elements must also be parameters on the corresponding model. The model's value for that parameter is then written into the element parameter member to initialize it. Then if the user specifies a value for the element parameter, that value is written over the initialized value. The parameters on the model may have an optional range string identified by the keyword RANGE. The characters following the keyword are interpreted as intervals where the format is the lower bound followed by a comma followed by the upper bound. The lower bound is inclusive if preceded by a square open bracket and is exclusive if preceded by an open parenthesis. The upper bound is inclusive if followed by a square close bracket and is exclusive if followed by a close parenthesis. The bound can be specified in the format of a signed floating point number with the extension that the character ">" is treated as positive infinity and the character "<" is treated as a very small positive number. The characters ">" and "<" may also be preceded by a minus sign. These range specifications are then used to check that what the user supplies is a reasonable value for the model. Default values for the parameters are not specified on the model data structure because the author believes that defaults should not be compiled into the executable code. Instead, the convention is used that when a model data structure is created, all of the parameters must be specified. The implementation of the routines that read in the circuit description allows models to reference other models and inherit their parameters. This makes it convenient to create libraries with default parameter values that are not compiled into the simulator.

If the keyword INITFUNC is present on the model data structure declaration, then the read routines will call the user-supplied optional initialization function. The keyword NETS indicates that the topological information for this device follows. The terminals of the device each have three pieces of information, the name of the terminal, the type of the terminal, and the directions of signal drive. The name of the terminal is used when elements are bound to wires by name, otherwise the position in the list of each terminal declaration corresponds to the position of that terminal on the input line. The type of the terminal is one of CIRCUIT, ELOGIC, LOGIC, or RTL. The direction of the signal drive is specified as one of INPUT, OUTPUT, and INANDOUT to indicate that the terminal only receives signals from the net, only drives signals onto the net, or is bidirectional, respectively.

The next part of the model is the evaluation function and optionally the initialization function. The electrical diode model evaluation function is declared on line 69 using the ELM_EVAL_DECLARE macro. This macro automatically declares the function to have the name of the model prefixed by the characters "Ele" and to return a value indicating the type of the signal driven. It also declares three arguments to the function the first of which is named "elm" and is a pointer to the element structure (struct ElmElcDiode in this case). The second argument is named "net" and is a pointer to the net that is requesting evaluation of the element and the third argument is named "result" and is a pointer to a signal union in which the result should be placed. The reader is directed to the file "signals.h" in Appendix A for the specific C language declarations of the signal types.

The variables "sig0" and "sig1" declared on line 71 are signal unions that will be used as buffers on lines 105 and 106 when the fanin nets are evaluated. The variable "diode" declared on line 72 is cast on line 87 to the pointer to the model for this element. The rest of the declared variables are used in the calculation of the device model of the diode. The lines 82-86 are used when debugging is enabled to perform run-time type checking using the type table that the type of the element evaluated is the one expected by this model. The lines 88-97 are used to set some of the local variables from the diode parameters. The lines 105 and 106 are where the fanin nets are evaluated. The macro NET_EVAL_FUNC is used to call the net evaluation routine with the proper arguments. The indices of the fanin and fanout lists are taken to be the order in which the corresponding terminals appear in the table of the model data structure declaration. In this case the first pointer (or zeroth index) is the terminal named "from" and the second pointer is the terminal named "to". The second argument of the macro is a compile-time constant that tells the net what type of signal is desired. The list of signal types is given in the file "signals.h". The third argument to the macro is the address of the signal union that is to hold the result that the net computes and the last argument to the macro is the time at which the net is requested to evaluate itself. The variable "SplCurTime" holds the current value of simulated time. Thus the voltage across the diode, vd, is computed on line 107 as the difference between to the values of the two terminal signals.

On lines 108 through 165, the linearization of the diode is performed as well as the calculation of the linearized capacitance. On lines 169-176, the Norton equivalent for the contribution of this element is computed. Note the statement on line 170 compares the net requesting evaluation with the nets on the fanin list to determine the sign of the branch current. On lines 177-179 the signal buffer of the net requesting evaluation of the diode is filled with the Norton equivalent for the diode and the type of the signal is returned on line 180.

The optional initialization function is declared using the MOD_INIT_DECLARE macro which declares one function argument with the name of the second macro argument to be a pointer to an element structure for the device indicated in the first macro argument. This function will be called for all elements and models of the device type. The name is constructed to be the prefix "Eli" followed by the name of the device being initialized. The variable named "diode" is declared on line 187 to permit convenient access to the model parameters later. The variables declared on lines 188-191 are specific to the calculations performed for the diode initialization. On line 193, the type of the argument is tested to see if it is the diode model type and if it is, the model pointer is initialized on line 194 and the code on lines 195-240 are executed to precalculate some of the variables used in the device model such as the critical voltage. Since no initialization is needed for the diode elements, there is no code outside the **if** statement range. The initialization function does not return a value, rather, the status of the routine is passed through the variable named "SetNoGo" which is set to TRUE on line 234 to indicate that the diode parameters are inconsistent.

The next step in producing the object code for the new model is to be sure that the "lint" program produces no errors for the model code. This can be done for the diode example by typing: "lint -u -I../h diode.mod". After the model code is "lint free", it is ready to be compiled. Be sure that there is a copy of the make file that is available in the "splice2/src/sys" directory named "makefile" in the current directory. That file has rules in it for creating the type tables by running the "stmake" program. After you are sure that there is a copy of that file or one with the same rules, type "make diode.o". This will run the "stmake" program and then the C compiler to produce the object code. You are now ready to use the new model for simulation. An example of a reference to the diode device would be:

```
.
.
.
model dmod ElcDiode : area=1, is=1e-14, n=1, tt=0, cjo=0, vj=1, m=.5, eg=1.11, $
     xti=3.0, fc=.5, bv=0, ibv=1e-3, couplefactor=0, uselimiting=1
d1 out GND dmod : area=2
.
.
.
```

To bind the model functions into the simulator, type:

```
splice2 -load diode.o <other options>
```

The option named "-load" will cause the program to run the loader and add your new function to the program when it executes.

APPENDIX E

Static Program Statistics

This appendix contains some statistics on the SPLICE2 program and its development. The following table summarizes the sizes of the program source code for SPLICE2.

Purpose	Lines	Semicolons
Include Files	2470	1067
Front End	11403	4802
Algorithmic Code	10579	5392
All of SPLICE2	29925	14992
Misc. Support Code	≈4000	≈2000

Include files typically contain data structure declarations and compile-time constant defines along with global function and variable declarations. The "front end" code consists of functions to read in and set up the data structures for simulation. Algorithmic code includes modeling and scheduling functions. Support code consists of the table generator program called "stmake" and other programs to read and write waveforms. The object code size of the SPLICE2 program with all of the default models loaded is:

Instructions	133120 Bytes
Data	147872 Bytes

APPENDIX

Table 3.1 Simple horizontal lines detector : Typical input file for circuit simulator PWLSPICE. The indices of nodes are coded as follow : the first number from the left stands for the type of the nodes in a cell, 1 for the state voltage node, 2 for the output voltage node; the second number from the left stands for the layer number; the third and fourth numbers stand for the rows and the fifth and sixth numbers stand for the columns, for instances, v(110102) means v_{x12}, and v(210304) means v_{y34}, and so on.

```
--------------------------------------------------------------------------------
      C110101             110101             0                   1e-09
      R110101             110101             0                   1000
      G1101010101   0   110101   210101   0          0.002
      G1101010102   0   110101   210102   0          0.001
      P210101             110101   210101    0                   modpwl1
      R210101             210101             0                   1
      C110102             110102             0                   1e-09
      R110102             110102             0                   1000
      G1101020101   0   110102   210101   0          0.001
      G1101020102   0   110102   210102   0          0.002
      G1101020103   0   110102   210103   0          0.001
      P210102             110102   210102    0                   modpwl1
      R210102             210102             0                   1
      C110103             110103             0                   1e-09
      R110103             110103             0                   1000
      G1101030102   0   110103   210102   0          0.001
      G1101030103   0   110103   210103   0          0.002
      G1101030104   0   110103   210104   0          0.001
      P210103             110103   210103    0                   modpwl1
      R210103             210103             0                   1
      C110104             110104             0                   1e-09
      R110104             110104             0                   1000
      G1101040103   0   110104   210103   0          0.001
      G1101040104   0   110104   210104   0          0.002
      P210104             110104   210104    0                   modpwl1
      R210104             210104             0                   1
      C110201             110201             0                   1e-09
      R110201             110201             0                   1000
      G1102010201   0   110201   210201   0          0.002
      G1102010202   0   110201   210202   0          0.001
      P210201             110201   210201    0                   modpwl1
      R210201             210201             0                   1
      C110202             110202             0                   1e-09
      R110202             110202             0                   1000
      G1102020201   0   110202   210201   0          0.001
--------------------------------------------------------------------------------
```

Table 3.1   continue

```
-------------------------------------------------------------------------------------------
G1102020202  0  110202  210202  0              0.002
G1102020203  0  110202  210203  0              0.001
P210202            110202  210202     0                  modpwl1
R210202            210202             0                  1
C110203            110203             0                  1e-09
R110203            110203             0                  1000
G1102030202  0  110203  210202  0              0.001
G1102030203  0  110203  210203  0              0.002
G1102030204  0  110203  210204  0              0.001
P210203            110203  210203     0                  modpwl1
R210203            210203             0                  1
C110204            110204             0                  1e-09
R110204            110204             0                  1000
G1102040203  0  110204  210203  0              0.001
G1102040204  0  110204  210204  0              0.002
P210204            110204  210204     0                  modpwl1
R210204            210204             0                  1
C110301            110301             0                  1e-09
R110301            110301             0                  1000
G1103010301  0  110301  210301  0              0.002
G1103010302  0  110301  210302  0              0.001
P210301            110301  210301     0                  modpwl1
R210301            210301             0                  1
C110302            110302             0                  1e-09
R110302            110302             0                  1000
G1103020301  0  110302  210301  0              0.001
G1103020302  0  110302  210302  0              0.002
G1103020303  0  110302  210303  0              0.001
P210302            110302  210302     0                  modpwl1
R210302            210302             0                  1
C110303            110303             0                  1e-09
R110303            110303             0                  1000
G1103030302  0  110303  210302  0              0.001
G1103030303  0  110303  210303  0              0.002
G1103030304  0  110303  210304  0              0.001
P210303            110303  210303     0                  modpwl1
R210303            210303             0                  1
C110304            110304             0                  1e-09
R110304            110304             0                  1000
G1103040303  0  110304  210303  0              0.001
G1103040304  0  110304  210304  0              0.002
P210304            110304  210304     0                  modpwl1
R210304            210304             0                  1
C110401            110401             0                  1e-09
R110401            110401             0                  1000
G1104010401  0  110401  210401  0              0.002
G1104010402  0  110401  210402  0              0.001
P210401            110401  210401     0                  modpwl1
R210401            210401             0                  1
C110402            110402             0                  1e-09
R110402            110402             0                  1000
G1104020401  0  110402  210401  0              0.001
G1104020402  0  110402  210402  0              0.002
G1104020403  0  110402  210403  0              0.001
P210402            110402  210402     0                  modpwl1
R210402            210402             0                  1
-------------------------------------------------------------------------------------------
```

Table 3.1   continue

```
--------------------------------------------------------------------------------------
          C110403                110403              0                 le-09
          R110403                110403              0                 1000
          G1104030402  0  110403  210402  0                   0.001
          G1104030403  0  110403  210403  0                   0.002
          G1104030404  0  110403  210404  0                   0.001
          P210403                110403  210403      0                 modpwl1
          R210403                210403              0                 1
          C110404                110404              0                 le-09
          R110404                110404              0                 1000
          G1104040403  0  110404  210403  0                   0.001
          G1104040404  0  110404  210404  0                   0.002
          P210404                110404  210404      0                 modpwl1
          R210404                210404              0                 1
          * This is a                PWL V           CCS model with a common node.
          .model  modpwl1 pwl  term = 3 nseg = 3
          + ap= 0,0 bp= 0,0,0,0 cp= 0,0,-0.5,0.5 alphap= 1,0,1,0  betap=-1,1
          .ic  v(110101)=-1  v(110102)=0.4  v(110103)=-0.8  v(110104)=-1
          +    v(110201)=-0.4  v(110202)=-1  v(110203)=-0.8  v(110204)=-0.6
          +    v(110301)=0.8  v(110302)=-0.4  v(110303)=0.8  v(110304)=1
          +    v(110401)=-0.8  v(110402)=-0.6  v(110403)=-0.8  v(110404)=-1
          .tran   0.1us    5us    UIC
          .print tran   v(210101) v(210102) v(210103) v(210104)
          .print tran   v(210201) v(210202) v(210203) v(210204)
          .print tran   v(210301) v(210302) v(210303) v(210304)
          .print tran   v(210401) v(210402) v(210403) v(210404)
          .end
--------------------------------------------------------------------------------------
```

Table 3.2 Simple horizontal lines detector: Transient analysis.
(The indices of the circuit nodes are the same as those in Table 3.1)

Index	TIME	v(210101)	v(210102)	v(210103)	v(210104)
0	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
1	1.000000E-09	-1.000000E+00	3.985972E-01	-8.014028E-01	-1.000000E+00
2	2.000000E-09	-1.000000E+00	3.971916E-01	-8.028084E-01	-1.000000E+00
3	4.000000E-09	-1.000000E+00	3.943747E-01	-8.056253E-01	-1.000000E+00
4	8.000000E-09	-1.000000E+00	3.887070E-01	-8.112930E-01	-1.000000E+00
5	1.600000E-08	-1.000000E+00	3.772346E-01	-8.227654E-01	-1.000000E+00
6	3.200000E-08	-1.000000E+00	3.537300E-01	-8.462700E-01	-1.000000E+00
7	6.400000E-08	-1.000000E+00	3.043898E-01	-8.956102E-01	-1.000000E+00
8	1.280000E-07	-1.000000E+00	1.957343E-01	-1.000000E+00	-1.000000E+00
9	2.280000E-07	-1.000000E+00	5.811578E-03	-1.000000E+00	-1.000000E+00
10	3.280000E-07	-1.000000E+00	-2.041030E-01	-1.000000E+00	-1.000000E+00
11	4.280000E-07	-1.000000E+00	-4.361138E-01	-1.000000E+00	-1.000000E+00
12	5.280000E-07	-1.000000E+00	-6.925469E-01	-1.000000E+00	-1.000000E+00
13	6.280000E-07	-1.000000E+00	-9.759729E-01	-1.000000E+00	-1.000000E+00
14	7.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
15	8.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
16	9.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
17	1.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
18	1.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
19	1.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
20	1.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
21	1.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
22	1.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
23	1.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
24	1.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
25	1.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
26	1.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
27	2.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
28	2.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
29	2.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
30	2.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
31	2.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
32	2.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
33	2.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
34	2.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
35	2.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
36	2.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
37	3.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
38	3.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
39	3.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
40	3.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
41	3.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
42	3.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
43	3.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
44	3.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
45	3.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
46	3.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
47	4.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
48	4.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
49	4.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
50	4.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
51	4.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
52	4.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
53	4.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
54	4.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
55	4.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
56	4.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
57	5.000000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00

Table 3.2 continue

Index	TIME	v(210201)	v(210202)	v(210203)	v(210204)
0	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
1	1.000000E-09	-4.014014E-01	-1.000000E+00	-8.024038E-01	-6.014038E-01
2	2.000000E-09	-4.028042E-01	-1.000000E+00	-8.048114E-01	-6.028114E-01
3	4.000000E-09	-4.056126E-01	-1.000000E+00	-8.096343E-01	-6.056343E-01
4	8.000000E-09	-4.112463E-01	-1.000000E+00	-8.193262E-01	-6.113262E-01
5	1.600000E-08	-4.225817E-01	-1.000000E+00	-8.388959E-01	-6.228959E-01
6	3.200000E-08	-4.455265E-01	-1.000000E+00	-8.787950E-01	-6.467950E-01
7	6.400000E-08	-4.925355E-01	-1.000000E+00	-9.617566E-01	-6.977566E-01
8	1.280000E-07	-5.912155E-01	-1.000000E+00	-1.000000E+00	-8.087408E-01
9	2.280000E-07	-7.587119E-01	-1.000000E+00	-1.000000E+00	-9.991345E-01
10	3.280000E-07	-9.438395E-01	-1.000000E+00	-1.000000E+00	-1.000000E+00
11	4.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
12	5.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
13	6.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
14	7.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
15	8.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
16	9.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
17	1.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
18	1.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
19	1.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
20	1.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
21	1.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
22	1.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
23	1.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
24	1.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
25	1.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
26	1.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
27	2.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
28	2.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
29	2.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
30	2.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
31	2.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
32	2.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
33	2.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
34	2.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
35	2.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
36	2.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
37	3.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
38	3.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
39	3.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
40	3.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
41	3.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
42	3.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
43	3.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
44	3.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
45	3.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
46	3.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
47	4.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
48	4.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
49	4.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
50	4.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
51	4.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
52	4.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
53	4.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
54	4.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
55	4.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
56	4.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
57	5.000000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00

Table 3.2 continue

Index	TIME	v(210301)	v(210302)	v(210303)	v(210304)
0	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
1	1.000000E-09	8.004016E-01	-3.987970E-01	0.000000E+00	0.000000E+00
2	2.000000E-09	8.008048E-01	-3.975910E-01	8.014026E-01	1.000000E+00
3	4.000000E-09	8.016145E-01	-3.951729E-01	8.028078E-01	1.000000E+00
4	8.000000E-09	8.032533E-01	-3.903003E-01	8.056235E-01	1.000000E+00
5	1.600000E-08	8.066099E-01	-3.804076E-01	8.112864E-01	1.000000E+00
6	3.200000E-08	8.136486E-01	-3.600176E-01	8.227396E-01	1.000000E+00
7	6.400000E-08	8.291051E-01	-3.166991E-01	8.461675E-01	1.000000E+00
8	1.280000E-07	8.662148E-01	-2.189430E-01	8.952019E-01	1.000000E+00
9	2.280000E-07	9.436895E-01	-4.146832E-02	1.000000E+00	1.000000E+00
10	3.280000E-07	1.000000E+00	1.617292E-01	1.000000E+00	1.000000E+00
11	4.280000E-07	1.000000E+00	3.892796E-01	1.000000E+00	1.000000E+00
12	5.280000E-07	1.000000E+00	6.407828E-01	1.000000E+00	1.000000E+00
13	6.280000E-07	1.000000E+00	9.187599E-01	1.000000E+00	1.000000E+00
14	7.280000E-07	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
15	8.280000E-07	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
16	9.280000E-07	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
17	1.028000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
18	1.128000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
19	1.228000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
20	1.328000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
21	1.428000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
22	1.528000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
23	1.628000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
24	1.728000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
25	1.828000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
26	1.928000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
27	2.028000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
28	2.128000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
29	2.228000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
30	2.328000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
31	2.428000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
32	2.528000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
33	2.628000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
34	2.728000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
35	2.828000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
36	2.928000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
37	3.028000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
38	3.128000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
39	3.228000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
40	3.328000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
41	3.428000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
42	3.528000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
43	3.628000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
44	3.728000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
45	3.828000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
46	3.928000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
47	4.028000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
48	4.128000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
49	4.228000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
50	4.328000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
51	4.428000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
52	4.528000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
53	4.628000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
54	4.728000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
55	4.828000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
56	4.928000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
57	5.000000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00

Table 3.2 continue

Index	TIME	v(210401)	v(210402)	v(210403)	v(210404)
0	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
1	1.000000E-09	-8.014036E-01	-6.022060E-01	-8.024046E-01	0.000000E+00
2	2.000000E-09	-8.028108E-01	-6.044181E-01	-8.048138E-01	-1.000000E+00
3	4.000000E-09	-8.056326E-01	-6.088542E-01	-8.096416E-01	-1.000000E+00
4	8.000000E-09	-8.113198E-01	-6.177994E-01	-8.193529E-01	-1.000000E+00
5	1.600000E-08	-8.228717E-01	-6.359847E-01	-8.390014E-01	-1.000000E+00
6	3.200000E-08	-8.467047E-01	-6.735635E-01	-8.792236E-01	-1.000000E+00
7	6.400000E-08	-8.974488E-01	-7.537920E-01	-9.635456E-01	-1.000000E+00
8	1.280000E-07	-1.000000E+00	-9.312657E-01	-1.000000E+00	-1.000000E+00
9	2.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
10	3.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
11	4.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
12	5.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
13	6.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
14	7.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
15	8.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
16	9.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
17	1.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
18	1.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
19	1.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
20	1.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
21	1.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
22	1.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
23	1.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
24	1.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
25	1.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
26	1.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
27	2.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
28	2.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
29	2.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
30	2.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
31	2.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
32	2.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
33	2.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
34	2.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
35	2.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
36	2.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
37	3.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
38	3.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
39	3.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
40	3.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
41	3.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
42	3.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
43	3.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
44	3.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
45	3.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
46	3.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
47	4.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
48	4.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
49	4.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
50	4.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
51	4.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
52	4.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
53	4.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
54	4.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
55	4.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
56	4.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
57	5.000000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00

FIGURE CAPTIONS:

Fig. 2.1: A two-dimensional cellular neural network. The circuit size is 4×4. The squares are the circuit units called *cells*. The links between the *cells* indicate that there are interactions between the linked *cells*.

Fig. 2.2: The neighborhood of cell $C(i,j)$ defined by (2.1) for $r = 1$, $r = 2$ and $r = 3$ respectively.

Fig. 2.3: An example of a *cell* circuit. C is a linear capacitor; R_x, R_u and R_y are linear resistors; I is an independent voltage source; $I_{xu}(i,j;k,l)$ and $I_{xy}(i,j;k,l)$ are linear voltage controlled current sources with the characteristics $I_{xy}(i,j;k,l) = A(i,j;k,l)v_{ykl}$ and $I_{xu}(i,j;k,l) = B(i,j;k,l)v_{ukl}$ for all $C(i,j) \in N(i,j)$; $I_{yx} = \frac{1}{R_y} f(v_{xij})$ is a piecewise-linear voltage controlled current source with its characteristic $f(\cdot)$ as shown in Fig. 2.4; E_{ij} is an independent voltage source.

Fig. 2.4: The characteristic of the nonlinear controlled source.

Fig. 2.5: The characteristic of the nonlinear resistor in the equivalent *cell* circuit.

Fig. 2.6: The steady state equivalent circuit of a *cell* in cellular neural networks.

Fig. 2.7: *Dynamic routes* and *equilibrium points* of the equivalent circuit for different values of $g(t)$.

Fig. 3.1: Input and output images for the simple example. (a) The input image to be processed; (b) the output image of the horizontal line detector; (c) the output image of the vertical line detector.

Fig. 3.2: A typical templet of an interactive cell operator. The unit used here is $10^{-3}\Omega^{-1}$.

Fig. 3.3: The cellular neural network description file of the simple horizontal line detector for the circuit simulation preprocessor CELL.

Fig. 3.4: The data file of CELL for the image in Fig. 3.1a.

Fig. 3.5: Templets for various interactive cell operators for noise removing cellular neural networks. The unit used here is $10^{-3}\Omega^{-1}$.

Fig. 3.6: The simulation result of a noise removing cellular neural network. Here $\sigma = 0.2$ and the interactive cell operator is defined by the templet in Fig. 3.5(a). (a) the upper left picture, is the input image; (b) the upper right picture, is the image at time step 10; (c) the lower left picture, is the image at time step 20; (d) the lower right picture, is the image at time step 30 (output image). (The color in this picture and all the following pictures is designed to stand for the gray levels of the pixels in the pictures, where

background = light blue;

$[-1.0, -\frac{7}{8}]$ = greenish blue; $(-\frac{7}{8}, -\frac{6}{8}]$ = blue green;

$(-\frac{6}{8}, -\frac{5}{8}]$ = bluish green; $(-\frac{5}{8}, -\frac{4}{8}]$ = green;

$(-\frac{4}{8}, -\frac{3}{8}]$ = yellowish green; $(-\frac{3}{8}, -\frac{2}{8}]$ = green yellow;

$(-\frac{2}{8}, -\frac{1}{8}]$ = greenish yellow; $(-\frac{1}{8}, 0.0]$ = yellow;

$(0.0, -\frac{1}{8}]$ = orangish yellow; $(\frac{1}{8}, \frac{2}{8}]$ = yellow orange;

$(\frac{2}{8}, \frac{3}{8}]$ = yellowish orange; $(\frac{3}{8}, \frac{4}{8}]$ = orange;
$(\frac{4}{8}, \frac{5}{8}]$ = reddish orange; $(\frac{5}{8}, \frac{6}{8}]$ = orange red;
$(\frac{6}{8}, \frac{7}{8}]$ = orangish red; $(\frac{7}{8}, 1.0]$ = red.)

Fig. 3.7: Simulation results of a noise removing cellular neural network. Here $\sigma = 0.4$ and the interactive cell operator is defined by the templet in Fig. 3.5(a). (a) the upper left picture, is the input image; (b) the upper right picture, is the image at time step 10; (c) the lower left picture, is the image at time step 20; (d) the lower right picture, is the image at time step 30 (output image).

Fig. 3.8: Simulation results of a noise removing cellular neural network. Here $\sigma = 0.2$ and the interactive cell operator is defined by the templet in Fig. 3.5(b). (a) the upper left picture, is the input image; (b) the upper right picture, is the image at time step 10; (c) the lower left picture, is the image at time step 20; (d) the lower right picture, is the image at time step 30 (output image).

Fig. 3.9: Simulation results of a noise removing cellular neural network. Here $\sigma = 0.4$ and the interactive cell operator is defined by the templet in Fig. 3.5(b). (a) the upper left picture, is the input image; (b) the upper right picture, is the image at time step 10; (c) the lower left picture, is the image at time step 20; (d) the lower right picture, is the image at time step 30 (output image).

Fig. 3.10: Simulation results of a noise removing cellular neural network. Here $\sigma = 0.2$ and the interactive cell operator is defined by the templet in Fig. 3.5(c). (a) the upper left picture, is the input image; (b) the upper middle picture, is the image at time step 10; (c) the upper right picture, is the image at time step 20; (d) the lower left picture, is the image at time

step 30; (e) the lower middle picture, is the image at time step 40; (f) the lower right picture, is the image at time step 57 (output image).

Fig. 3.11: Simulation results of a noise removing cellular neural network. Here $\sigma = 0.4$ and the interactive cell operator is defined by the templet in Fig. 3.5(c). (a) the upper left picture, is the input image; (b) the upper middle picture, is the image at time step 10; (c) the upper right picture, is the image at time step 20; (d) the lower left picture, is the image at time step 30; (e) the lower middle picture, is the image at time step 40; (f) the lower right picture, is the image at time step 58 (output image).

Fig. 3.12: Simulation results of a noise removing cellular neural network. Here the interactive cell operator is defined by the templet in Fig. 3.5(a). (a) the upper left picture, is the input image; (b) the upper middle picture, is the image at time step 10; (c) the upper right picture, is the image at time step 20; (d) the lower left picture, is the image at time step 30; (e) the lower middle picture, is the image at time step 40; (f) the lower right picture, is the image at time step 57 (output image).

Fig. 3.13: Simulation results of a noise removing cellular neural network. Here the interactive cell operator is defined by the templet in Fig. 3.5(d). (a) the upper left picture, is the input image; (b) the upper middle picture, is the image at time step 10; (c) the upper right picture, is the image at time step 20; (d) the lower left picture, is the image at time step 30; (e) the lower middle picture, is the image at time step 40; (f) the lower right picture, is the image at time step 57 (output image).

Fig. 3.14: Simulation results of a noise removing cellular neural network. Here $\sigma = 0.6$ and the interactive cell operator is defined by the templet in Fig. 3.5(b). (a) the upper left picture, is the input image; (b) the upper right picture, is the image at time step 10; (c) the lower left picture, is the image at time step 20; (d) the lower right picture, is the image at time

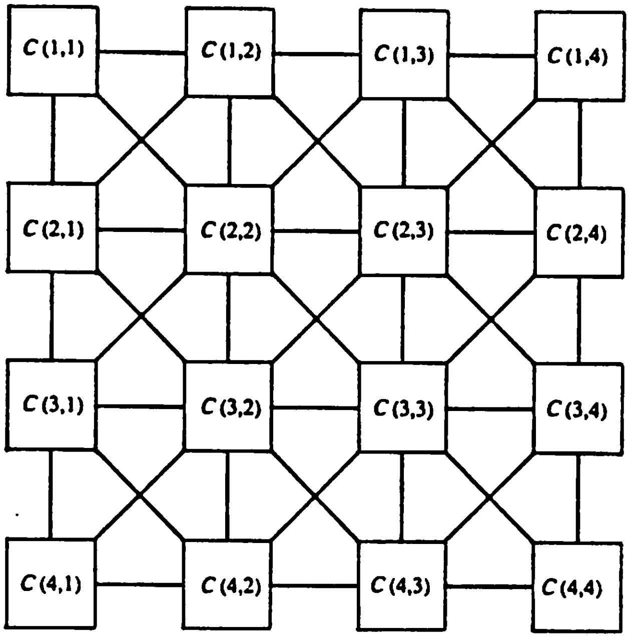
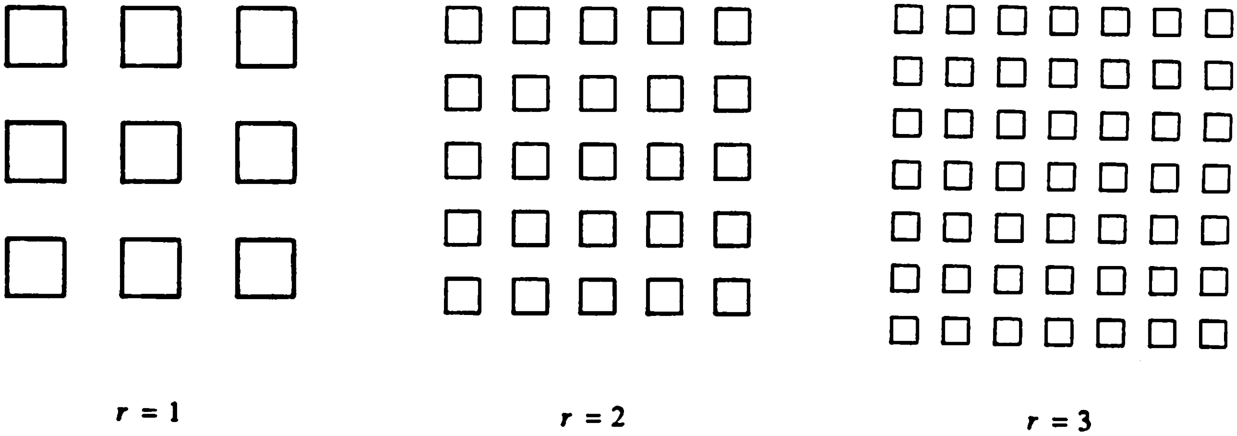
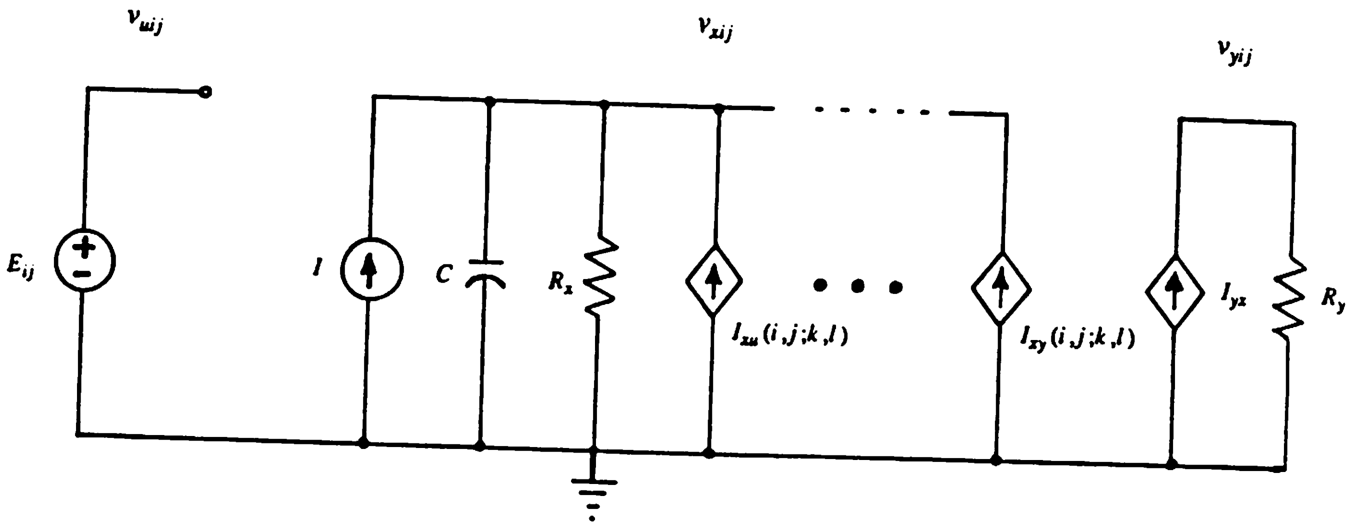


Fig. 2.1



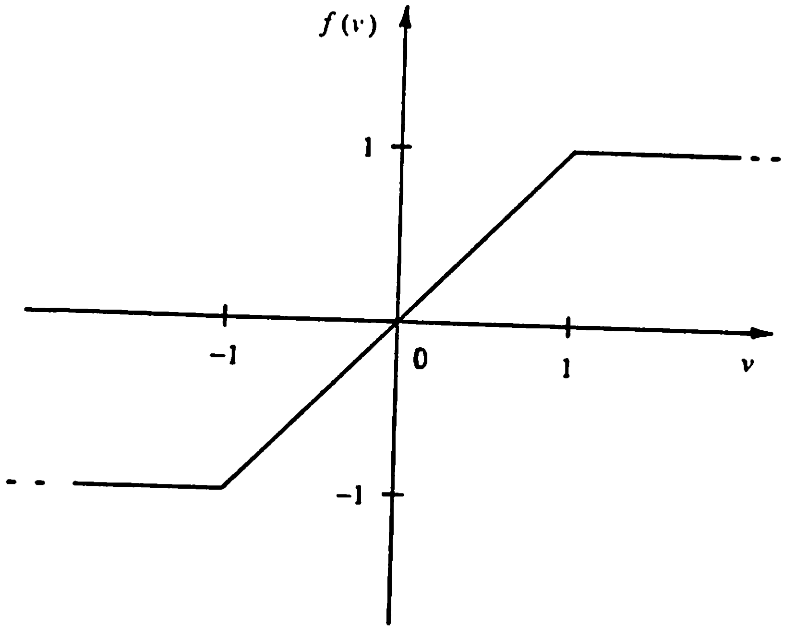
$r = 1$ $r = 2$ $r = 3$

Fig. 2.2



$I_{xu}(i,j;k,l) = B(i,j;k,l)v_{ukl}$, $I_{xy}(i,j;k,l) = A(i,j;k,l)v_{ykl}$, $I_{yx} = 0.5\left[\,|v_{xij}+1| - |v_{xij}-1|\,\right]$.

Fig. 2.3



$$f(v) = \frac{1}{2}\left[\,|v+1| - |v-1|\,\right]$$

Fig. 2.4

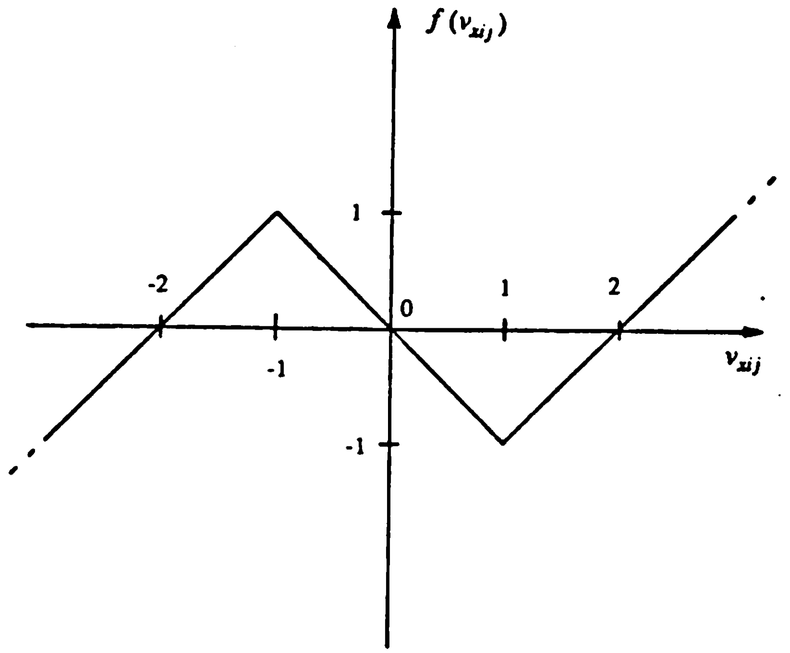


Fig. 2.5

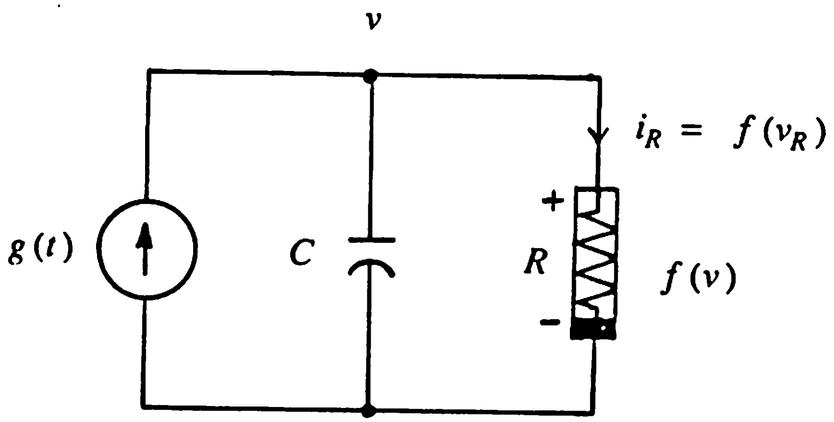


Fig. 2.6

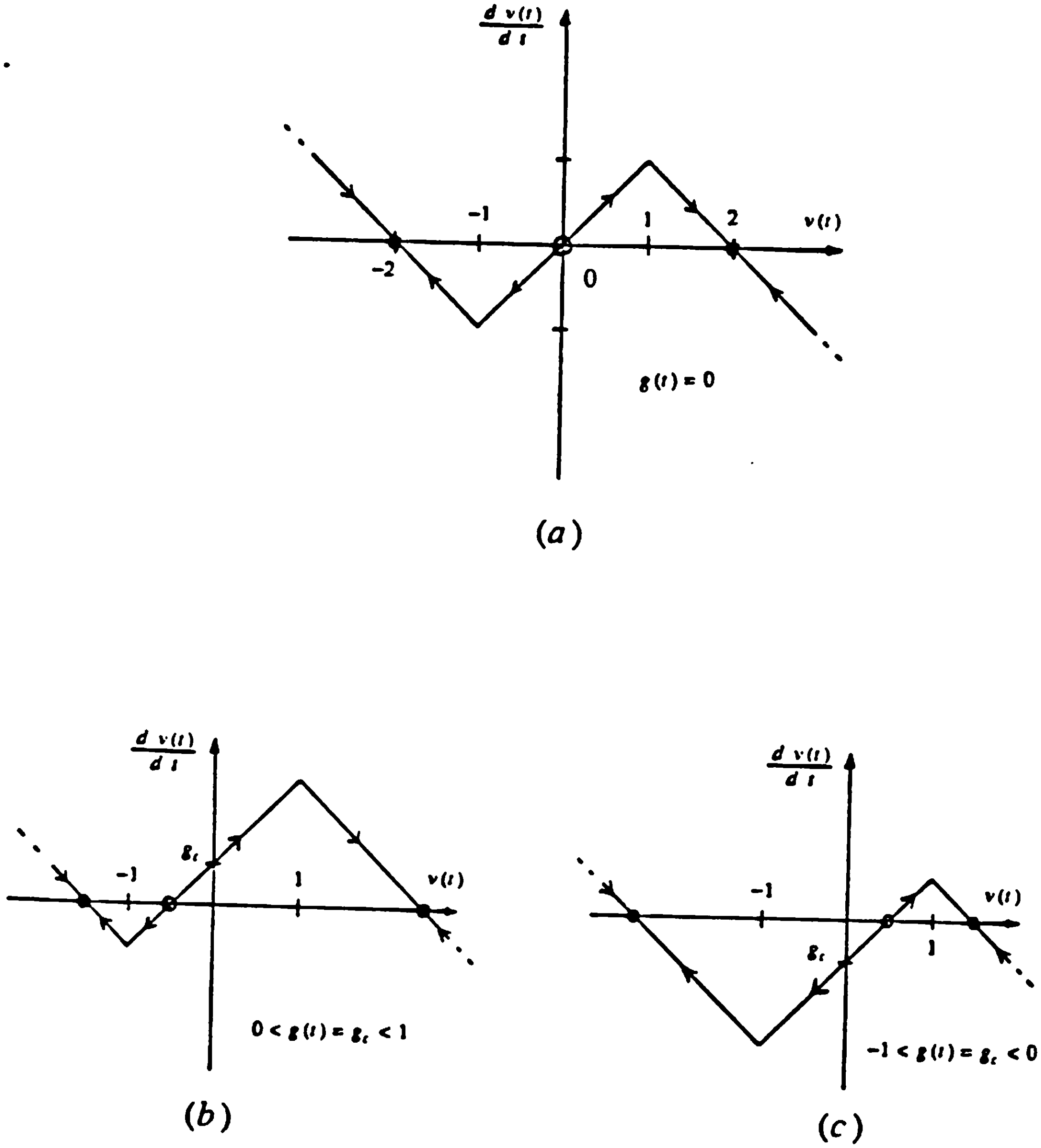
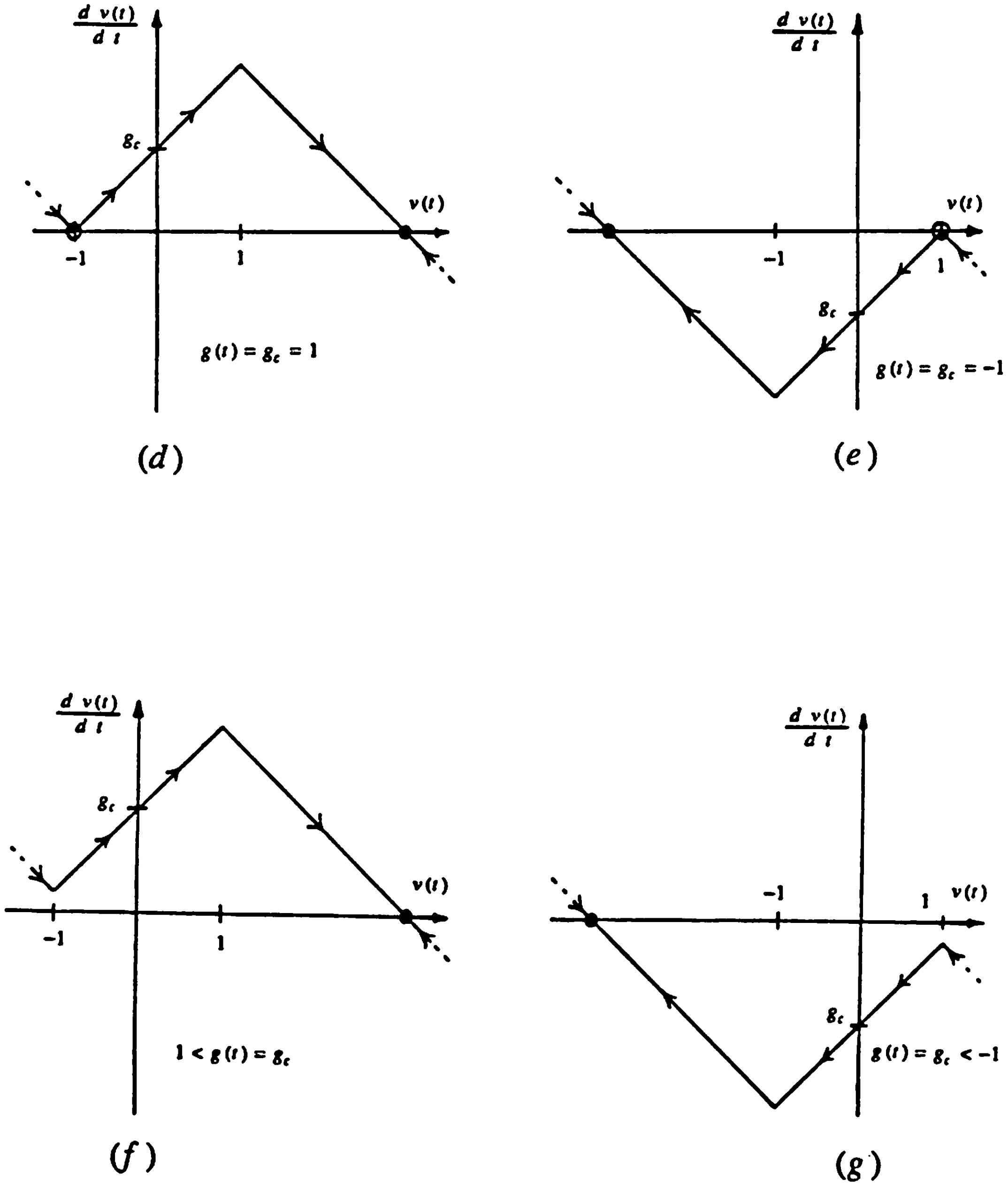

d v(t) / d t
-1
1
2
v(t)
-2
0
g(t) = 0
(a)
d v(t) / d t
g_c
-1
1
v(t)
0 < g(t) = g_c < 1
(b)
d v(t) / d t
-1
v(t)
g_c
1
-1 < g(t) = g_c < 0
(c)

Fig. 2.7 (a, b, c)



(d)

(e)

(f)

(g)

Fig. 2.7 (d, e, f, g)

-1.0	0.4	-0.8	-1.0
-0.4	-1.0	-0.8	-0.6
0.8	-0.4	0.8	1.0
-0.8	-0.6	-0.8	-1.0

(a)

-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
1.0	1.0	1.0	1.0
-1.0	-1.0	-1.0	-1.0

(b)

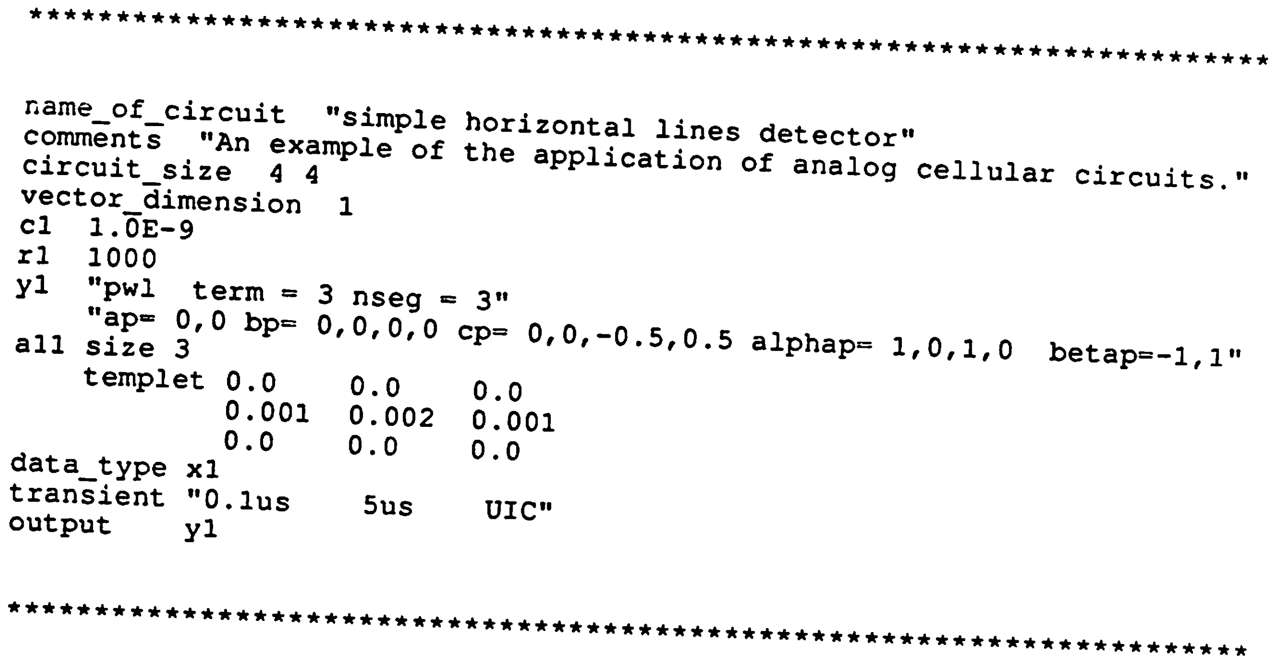
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0

(c)

Fig. 3.1

0.0	0.0	0.0
1.0	2.0	1.0
0.0	0.0	0.0

Fig. 3.2

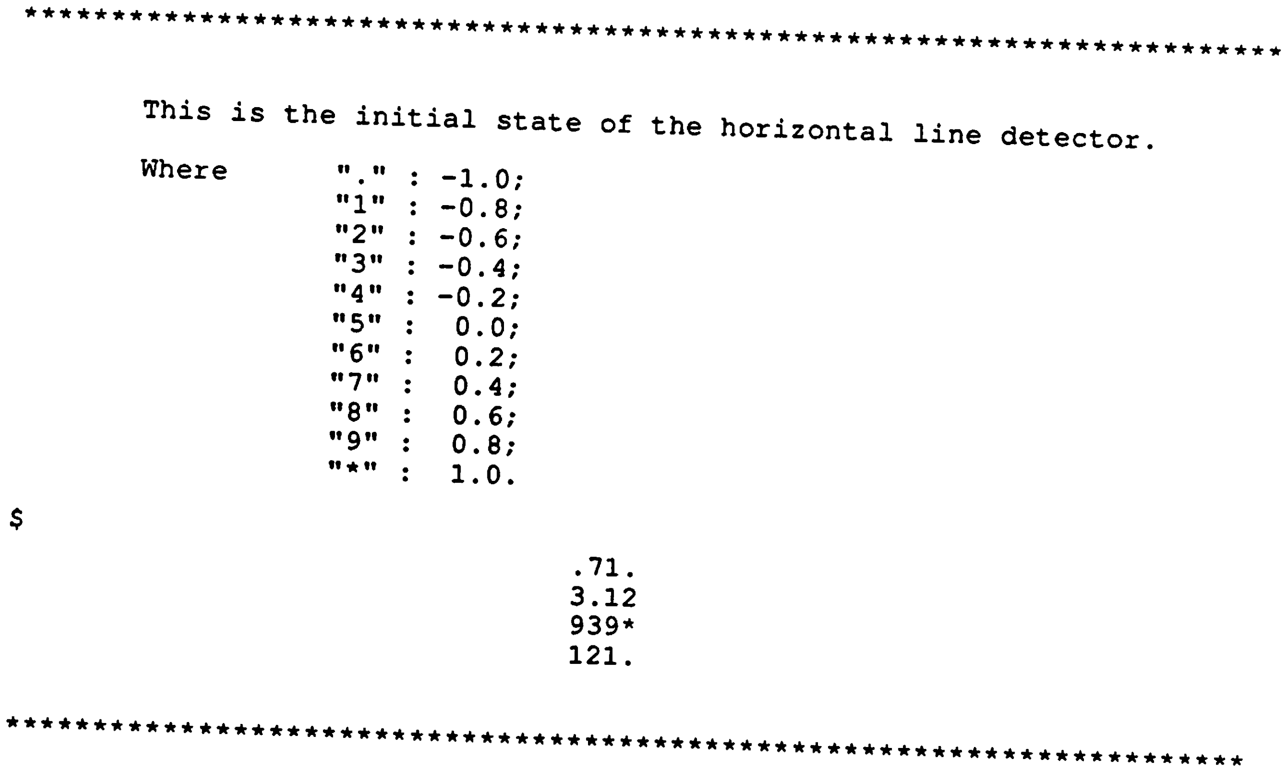


```
**********************************************************************

name_of_circuit  "simple horizontal lines detector"
comments  "An example of the application of analog cellular circuits."
circuit_size  4 4
vector_dimension  1
c1  1.0E-9
r1  1000
y1  "pwl  term = 3 nseg = 3"
    "ap= 0,0 bp= 0,0,0,0 cp= 0,0,-0.5,0.5 alphap= 1,0,1,0  betap=-1,1"
all size 3
    templet 0.0     0.0    0.0
            0.001   0.002  0.001
            0.0     0.0    0.0
data_type x1
transient "0.1us    5us    UIC"
output    y1

**********************************************************************
```

Fig. 3.3



```
**********************************************************************

        This is the initial state of the horizontal line detector.

        Where       "." : -1.0;
                    "1" : -0.8;
                    "2" : -0.6;
                    "3" : -0.4;
                    "4" : -0.2;
                    "5" :  0.0;
                    "6" :  0.2;
                    "7" :  0.4;
                    "8" :  0.6;
                    "9" :  0.8;
                    "*" :  1.0.

$
                            .71.
                            3.12
                            939*
                            121.

**********************************************************************
```

Fig. 3.4

0.0	1.0	0.0
1.0	2.0	1.0
0.0	1.0	0.0

(a)

0.0	1.0	0.0
1.0	4.0	1.0
0.0	1.0	0.0

(b)

0.5	1.0	0.5
1.0	4.0	1.0
0.5	1.0	0.5

(c)

0.0	0.0	0.0
0.0	4.0	0.0
0.0	0.0	0.0

(d)

Fig. 3.5

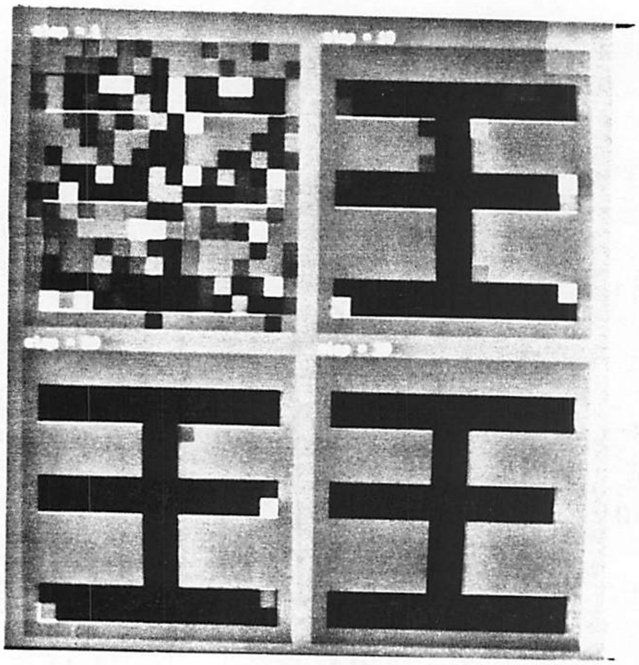


Fig. 3.6

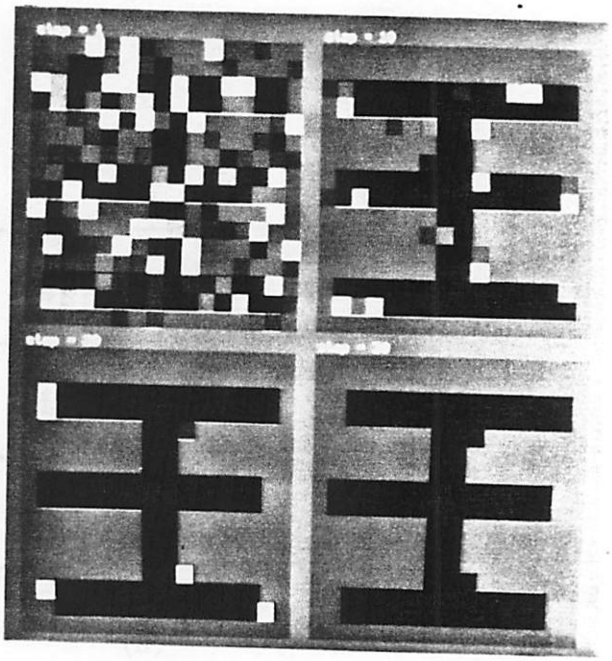


Fig. 3.7

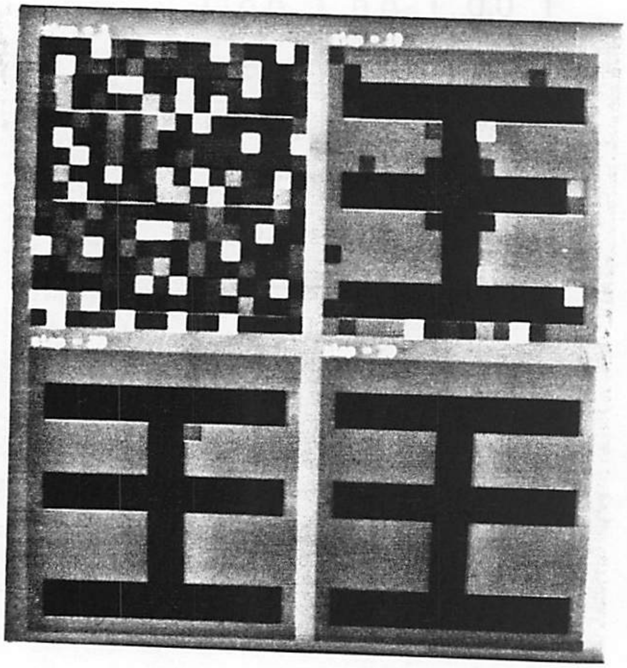


Fig. 3.8

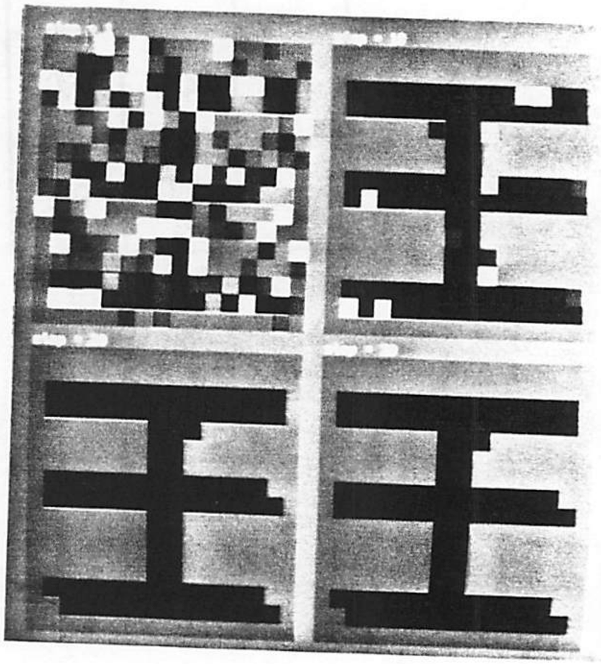


Fig. 3.9

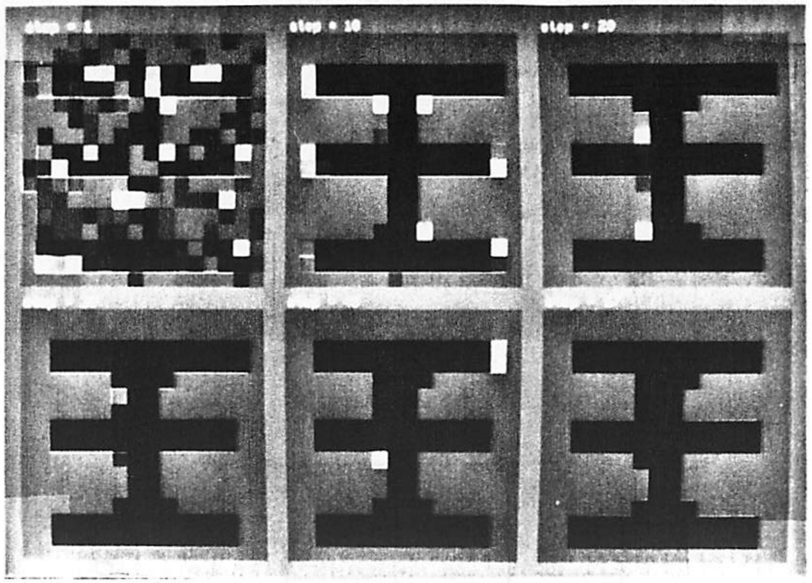


Fig. 3.10

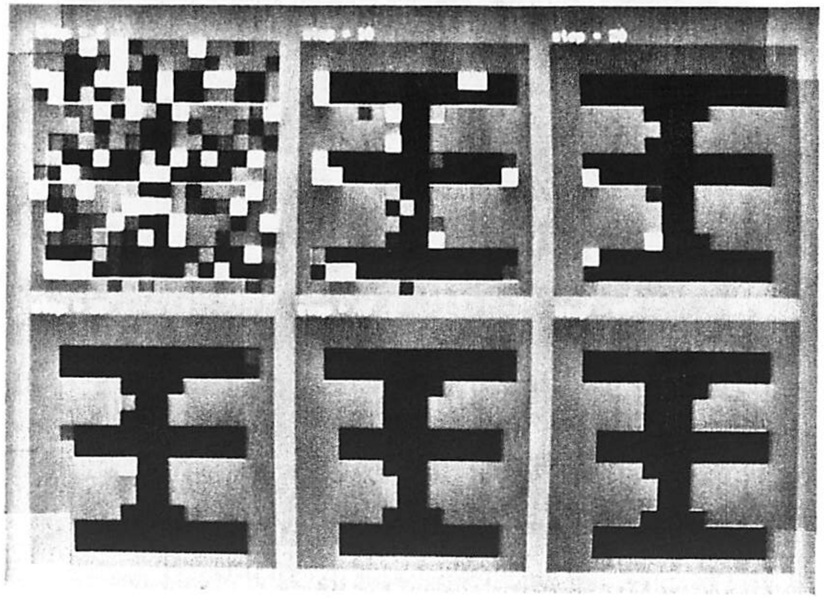


Fig. 3.11

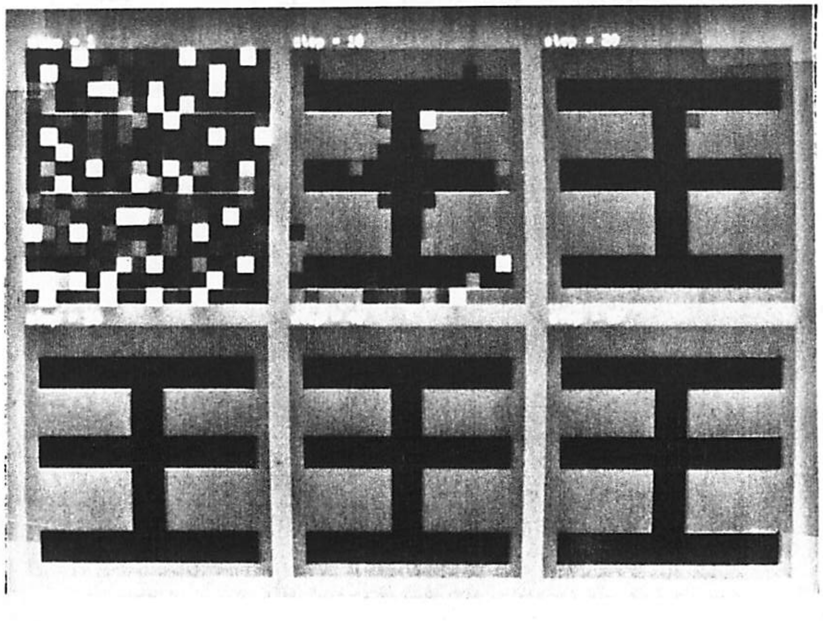


Fig. 3.12

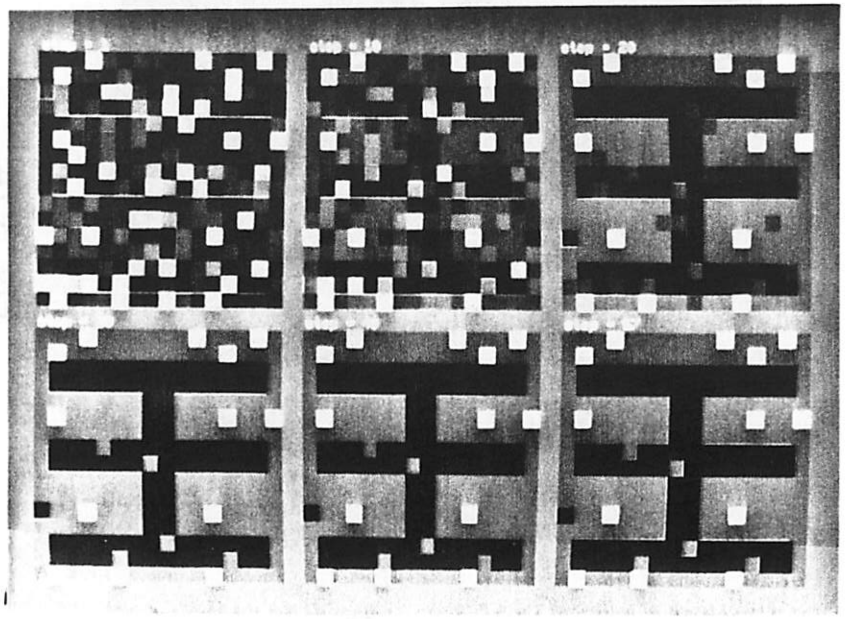


Fig. 3.13

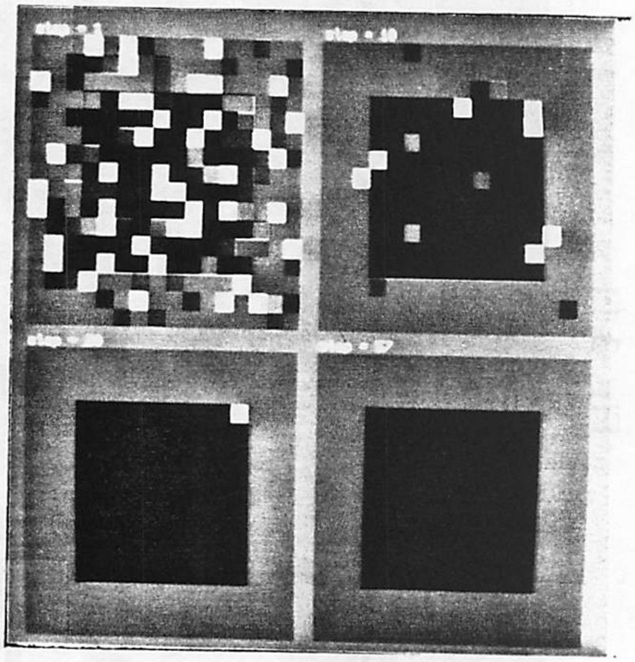


Fig. 3.14

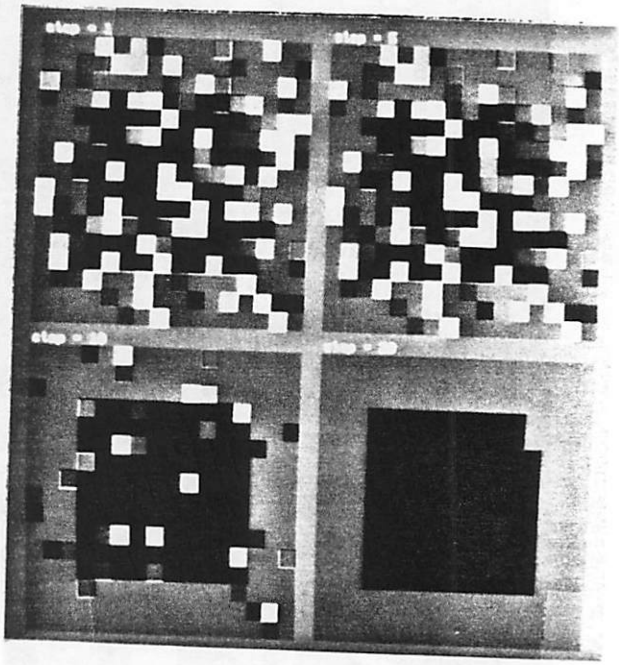


Fig. 3.15

0.0	-0.5	0.0
-0.5	2.0	-0.5
0.0	-0.5	0.0

Fig. 3.16

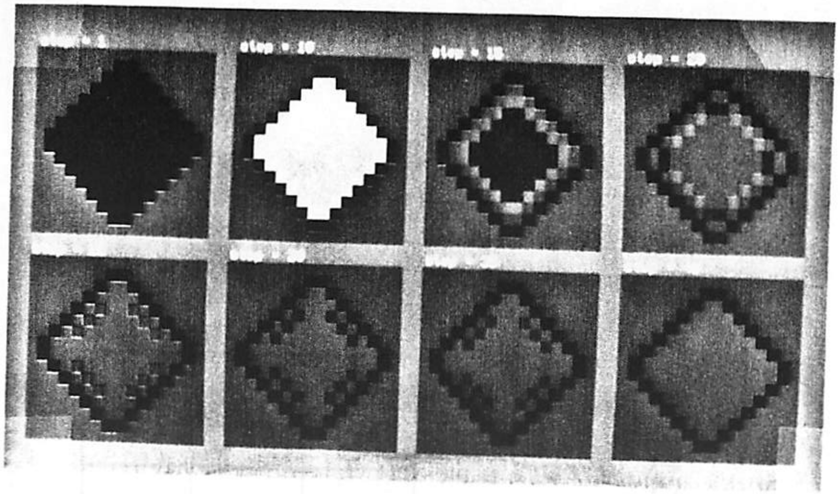


Fig. 3.17

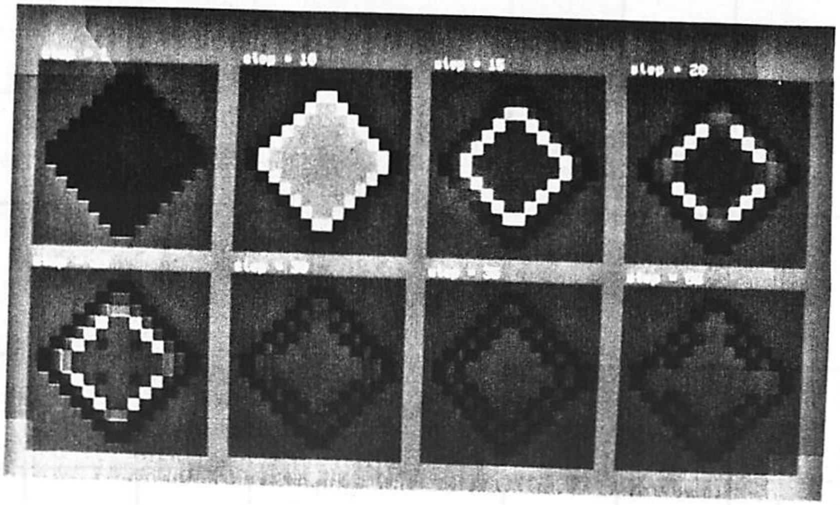


Fig. 3.18

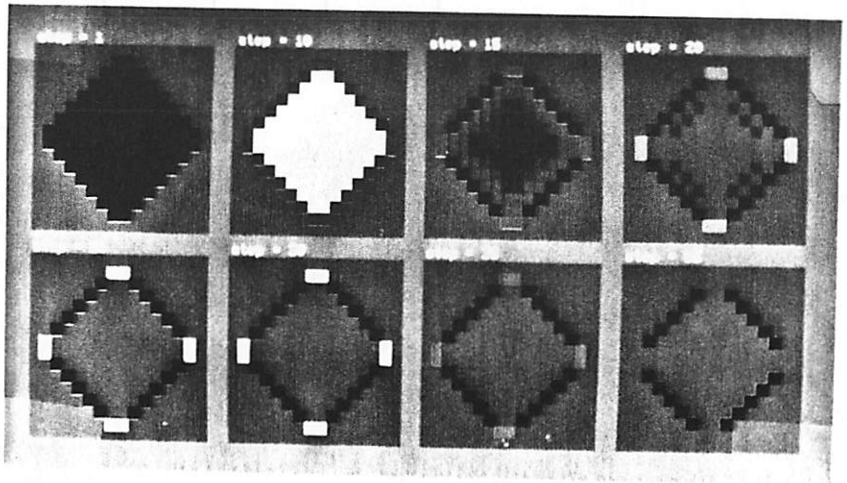


Fig. 3.19

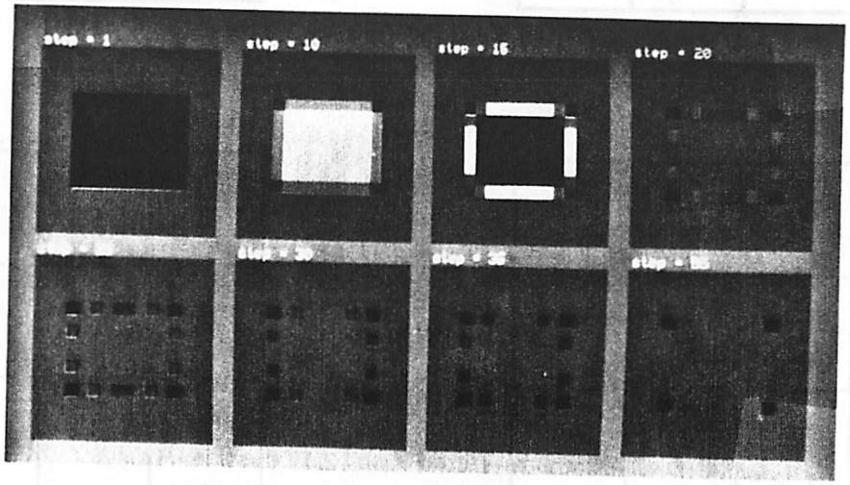


Fig. 3.20

-1	-1	-1
-1	-1	-1
-1	-1	-1

(a)

1	1	1
1	1	1
1	1	1

(b)

-1	-1	-1
-1	-1	-1
1	1	1

(c)

-1	-1	-1
1	1	1
1	1	1

(d)

-1	-1	-1
-1	-1	-1
-1	-1	1

(e)

-1	-1	-1
-1	1	1
-1	1	1

(f)

Fig. 3.21

0.0	-1.0	0.0
-1.0	4.0	-1.0
0.0	-1.0	0.0

Fig. 3.22

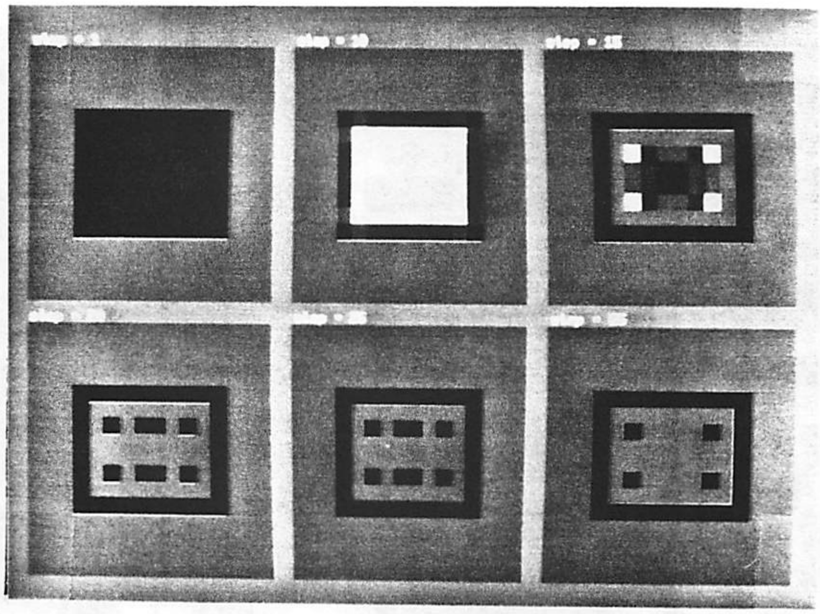


Fig. 3.23

0.0	0.0	0.0
0.0	2.0	0.0
0.0	0.0	0.0

(a)

-0.25	-0.25	-0.25
-0.25	2.0	-0.25
-0.25	-0.25	-0.25

(b)

Fig. 3.24

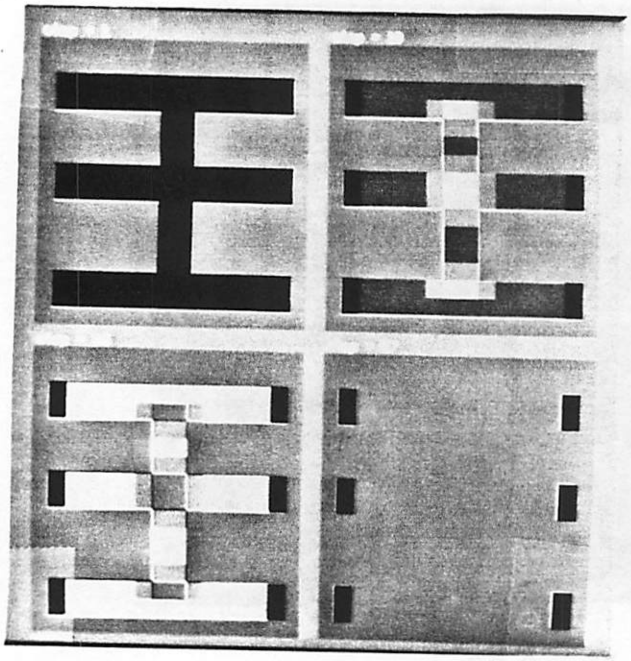


Fig. 3.29

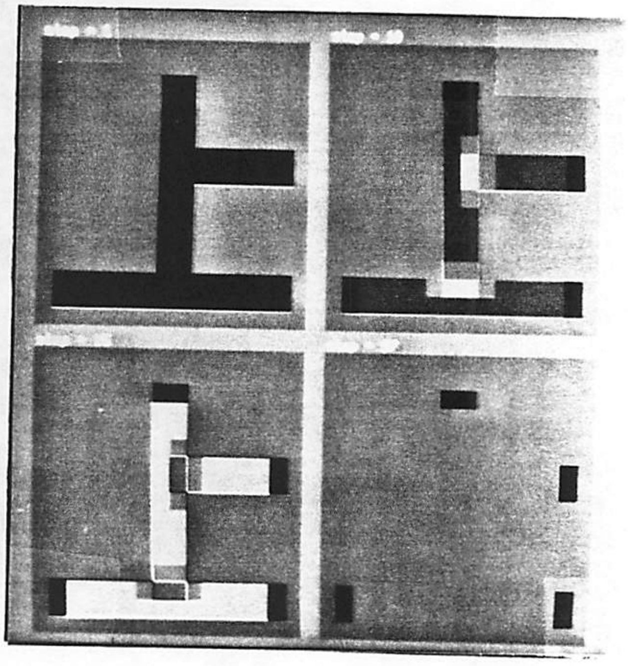


Fig. 3.30

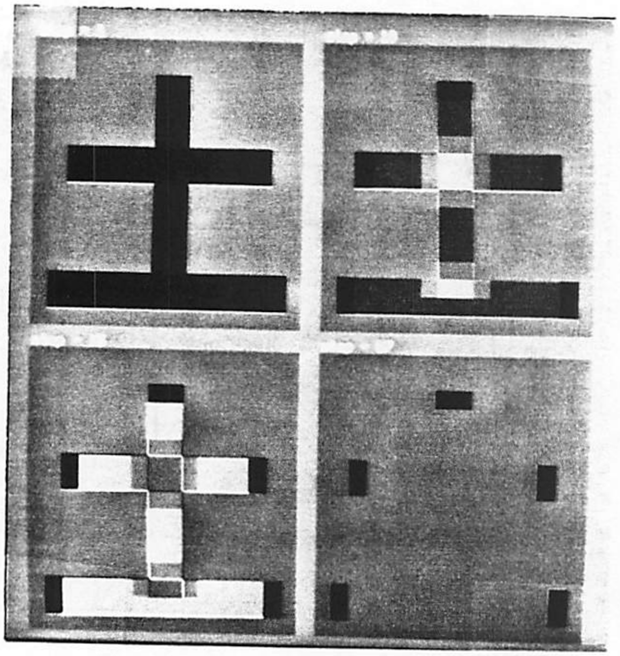


Fig. 3.31

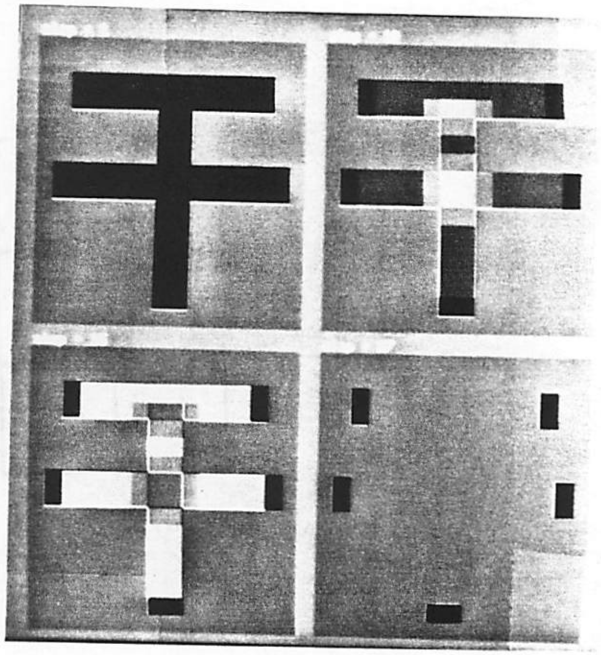


Fig. 3.32

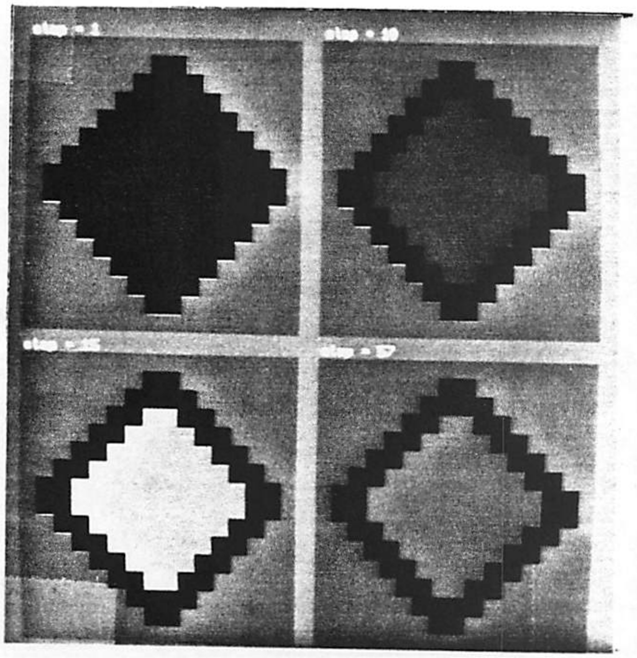


Fig. 3.25

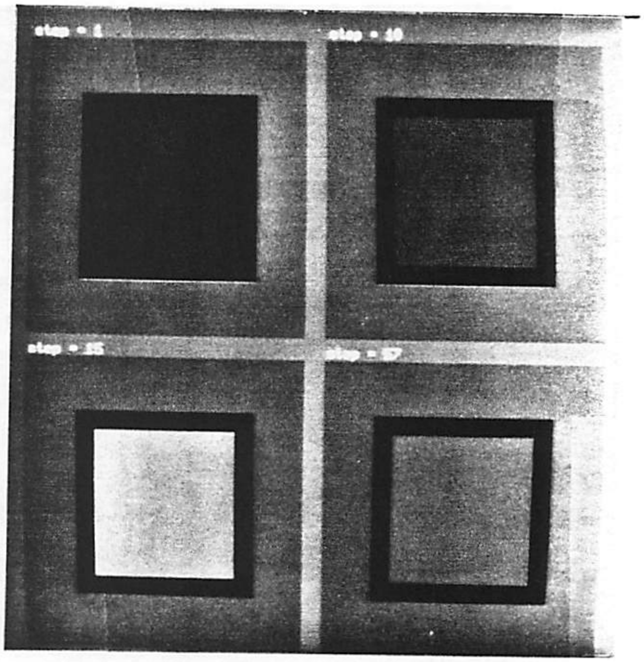


Fig. 3.26

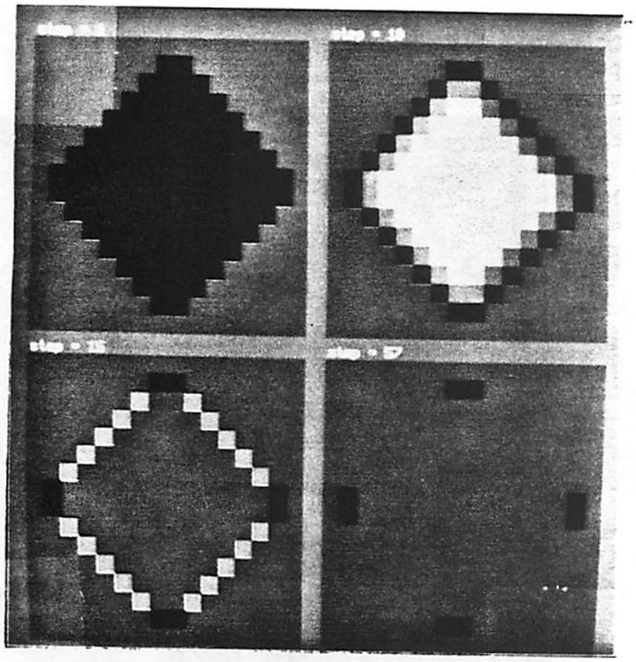


Fig. 3.27

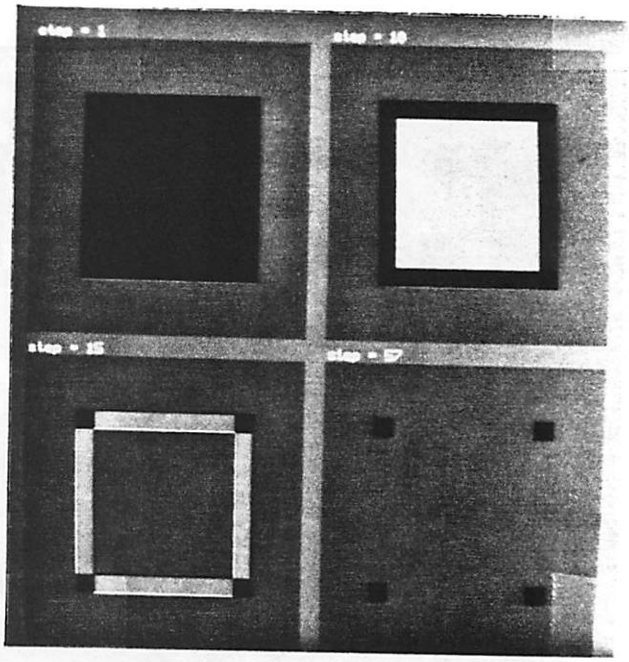


Fig. 3.28

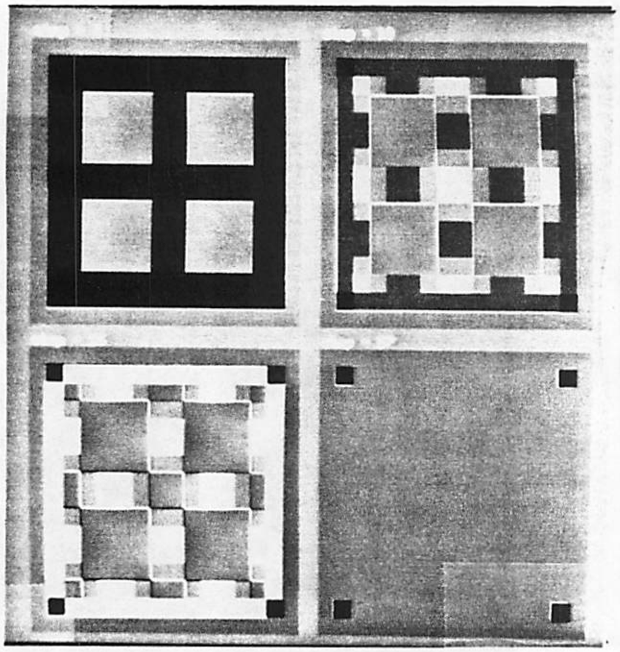


Fig. 3.33

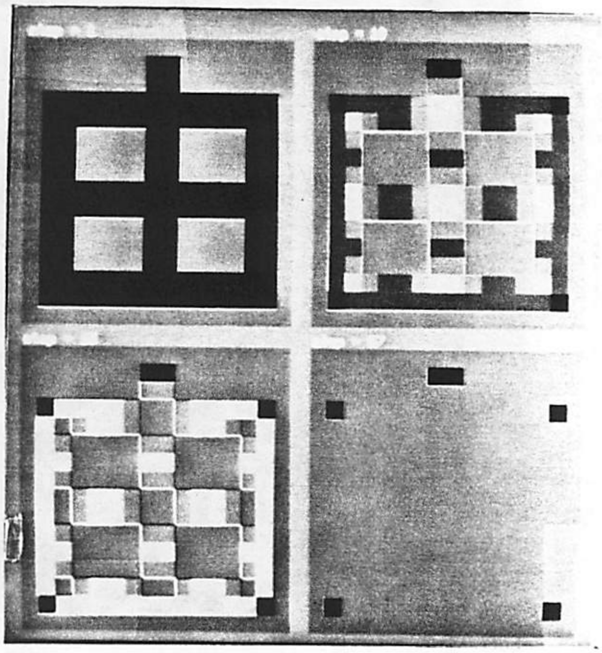


Fig. 3.34

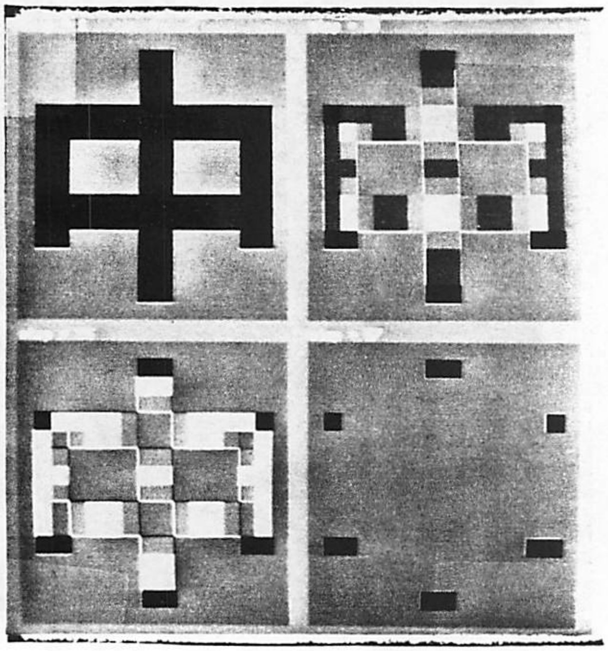


Fig. 3.35

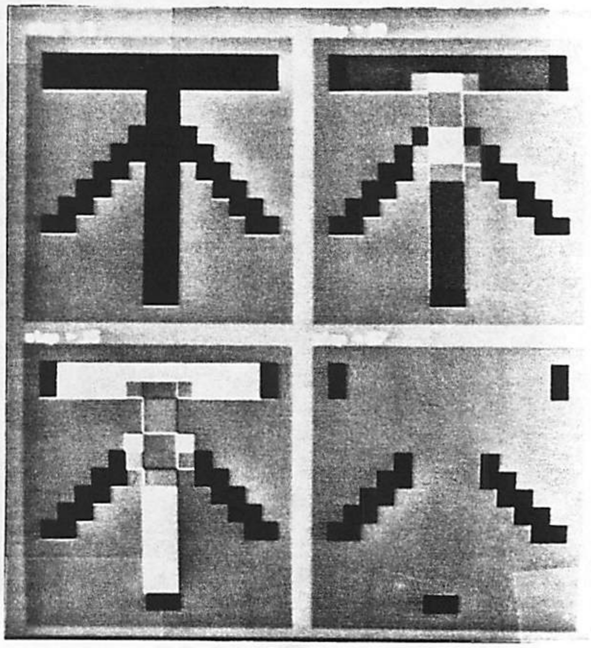


Fig. 3.36

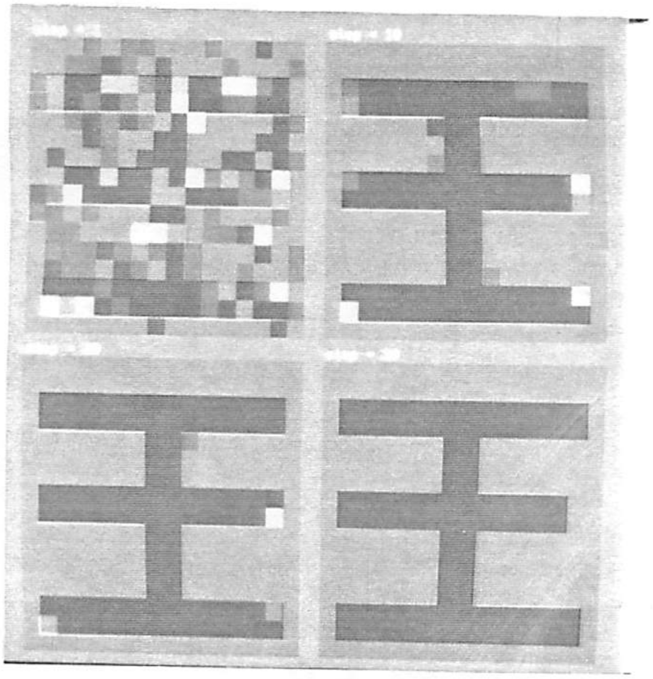


Fig. 3.6

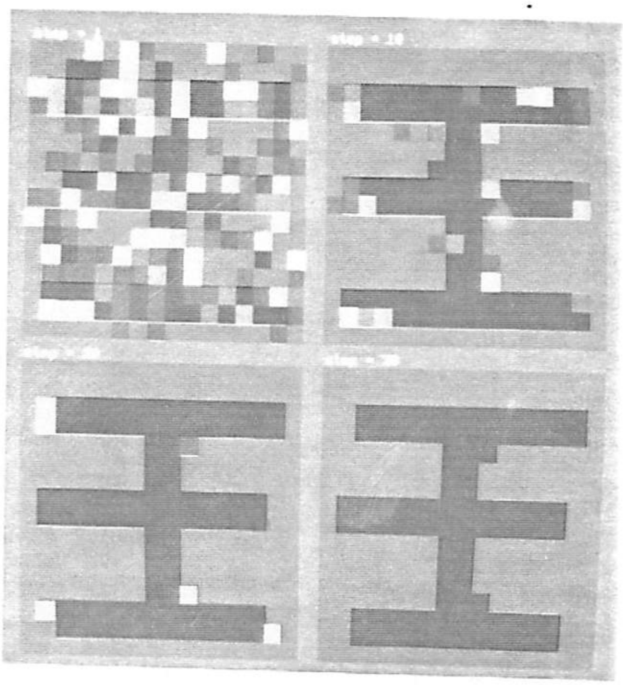


Fig. 3.7

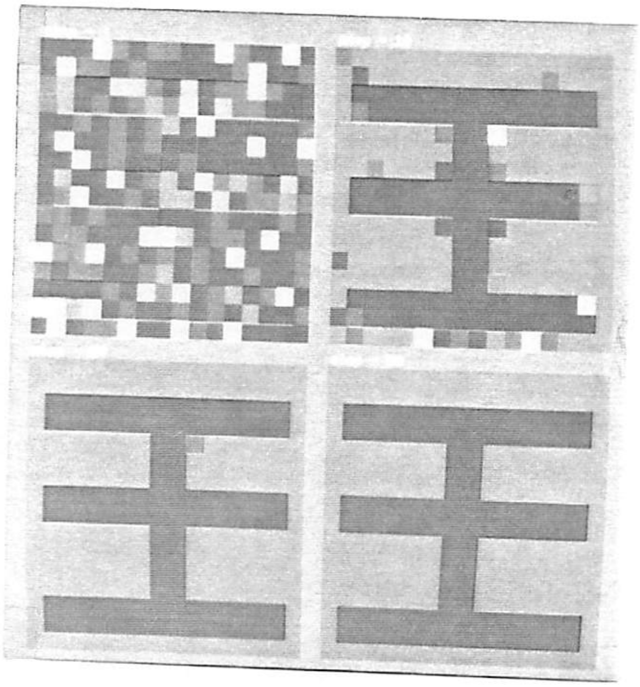


Fig. 3.8

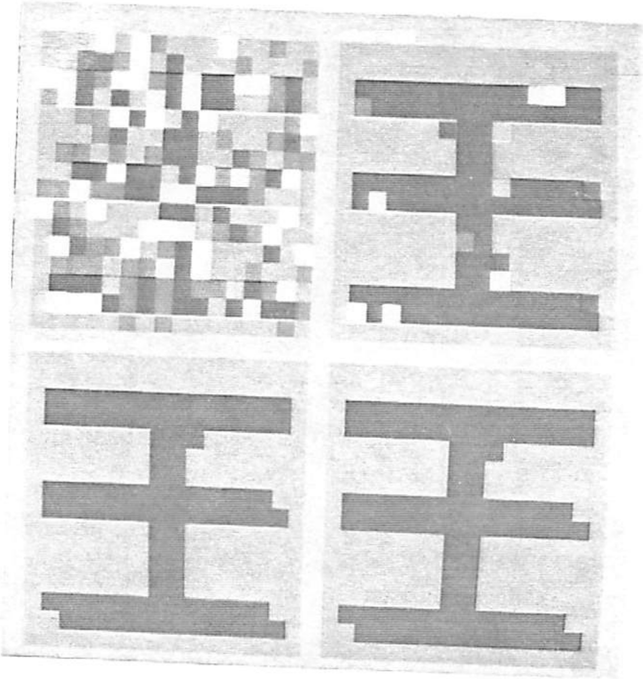


Fig. 3.9

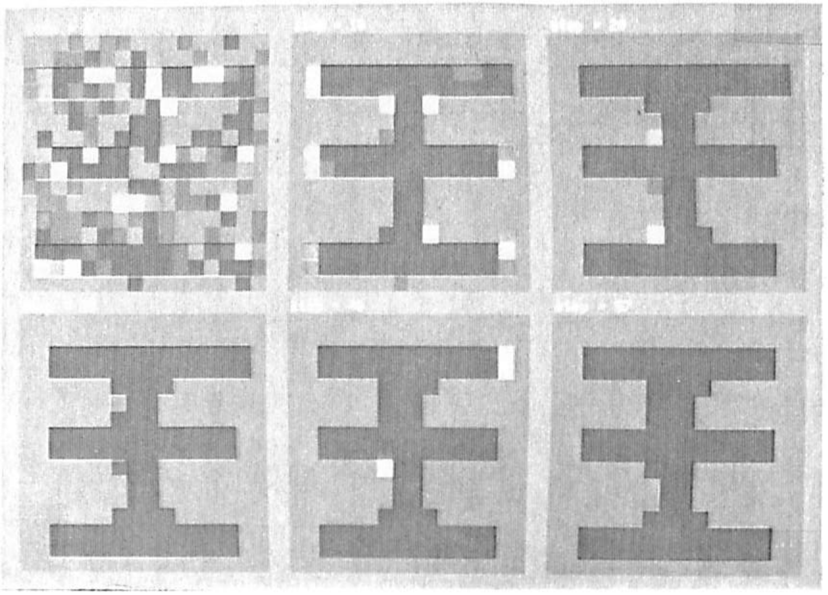


Fig. 3.10

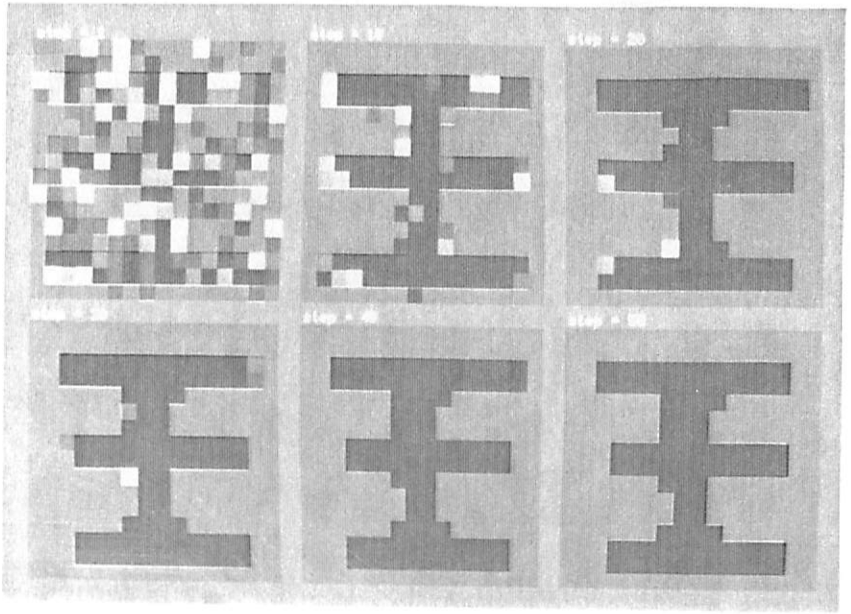


Fig. 3.11

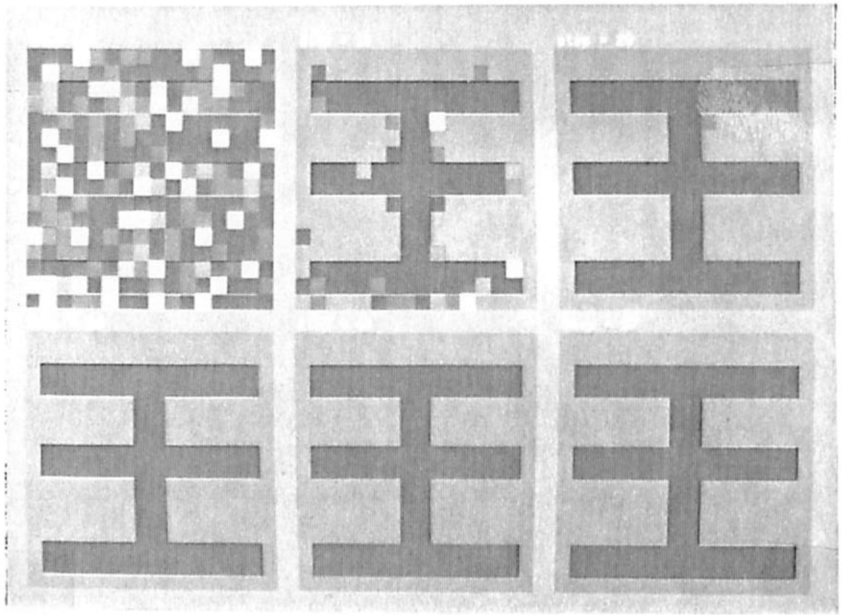


Fig. 3.12

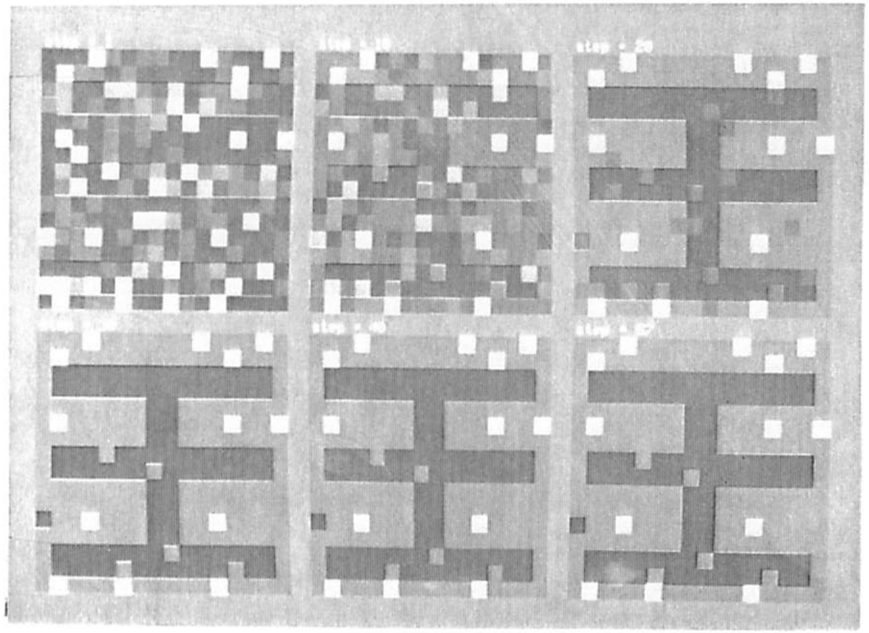


Fig. 3.13

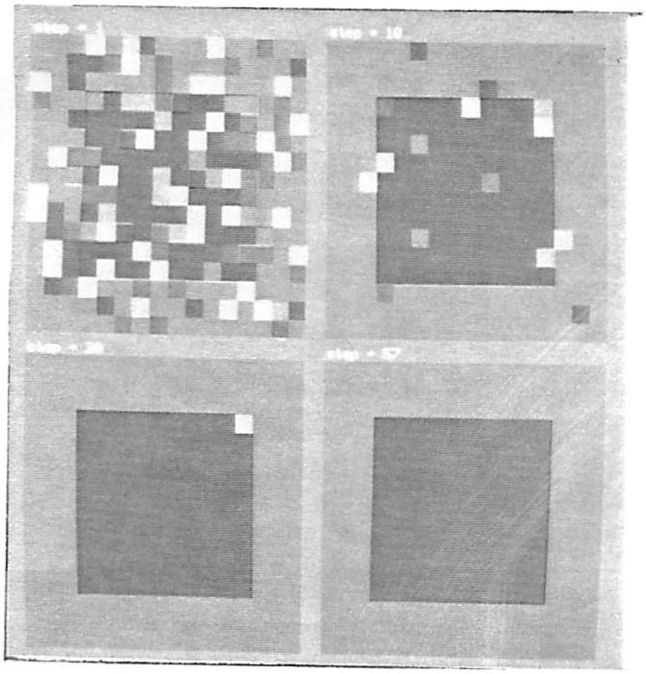


Fig. 3.14

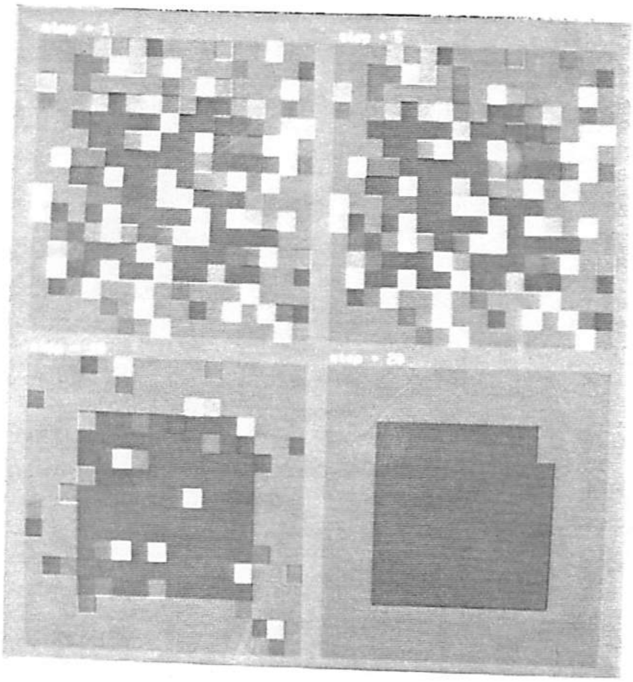


Fig. 3.15

0.0	-0.5	0.0
-0.5	2.0	-0.5
0.0	-0.5	0.0

Fig. 3.16

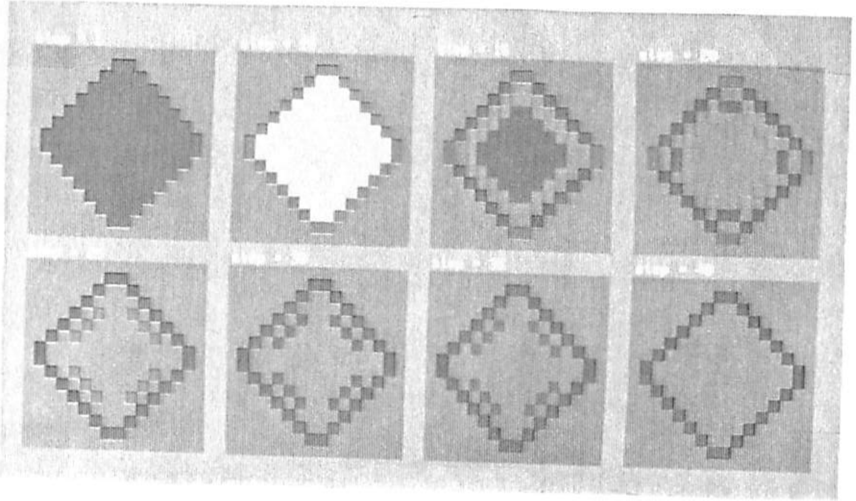


Fig. 3.17

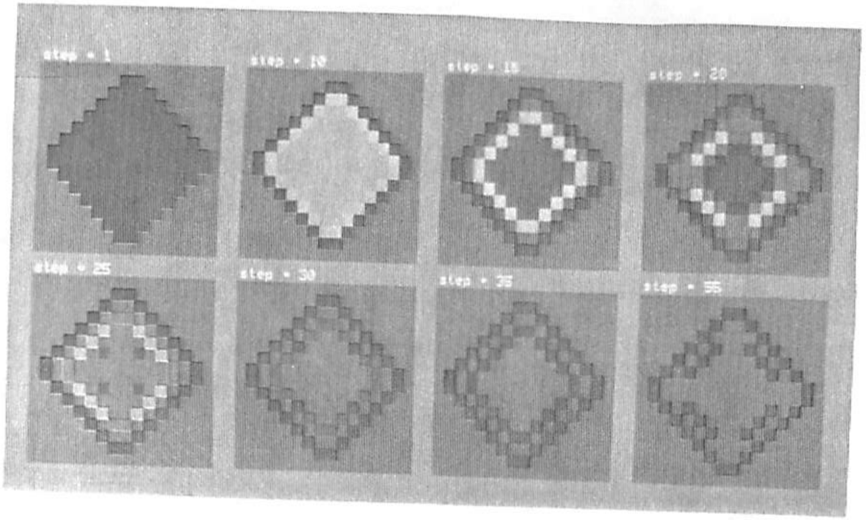


Fig. 3.18

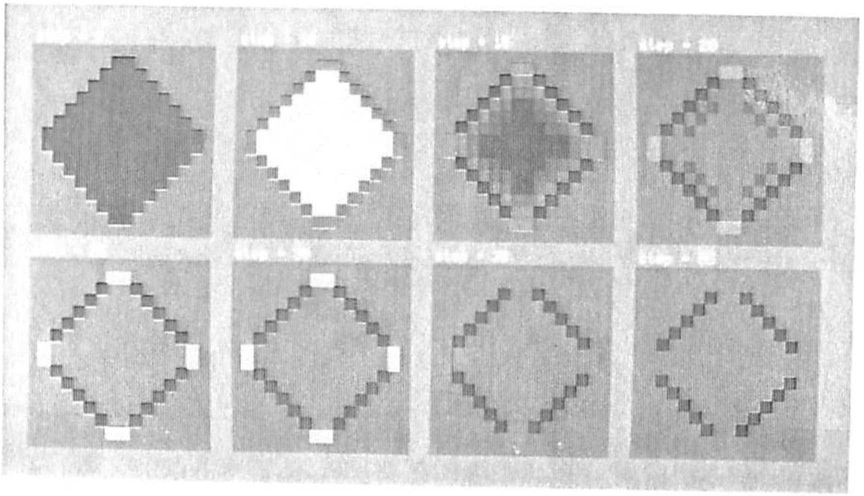


Fig. 3.19

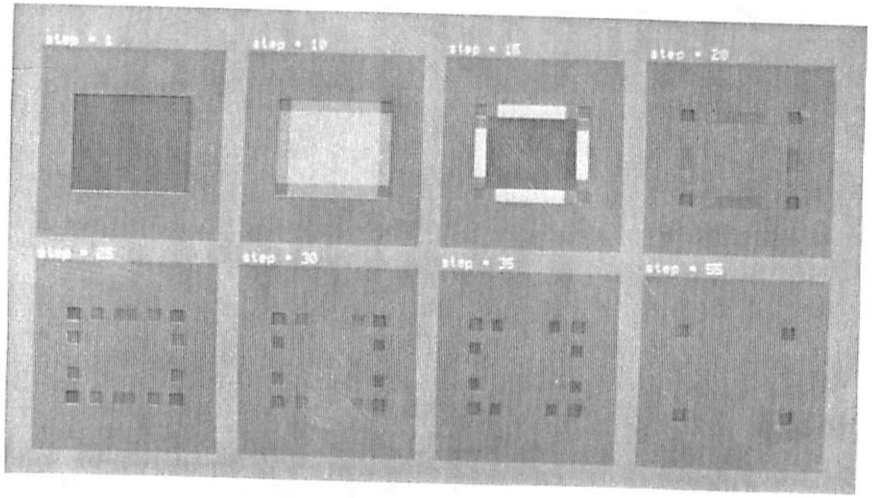


Fig. 3.20

-1	-1	-1
-1	-1	-1
-1	-1	-1

(a)

1	1	1
1	1	1
1	1	1

(b)

-1	-1	-1
-1	-1	-1
1	1	1

(c)

-1	-1	-1
1	1	1
1	1	1

(d)

-1	-1	-1
-1	-1	-1
-1	-1	1

(e)

-1	-1	-1
-1	1	1
-1	1	1

(f)

Fig. 3.21

0.0	-1.0	0.0
-1.0	4.0	-1.0
0.0	-1.0	0.0

Fig. 3.22

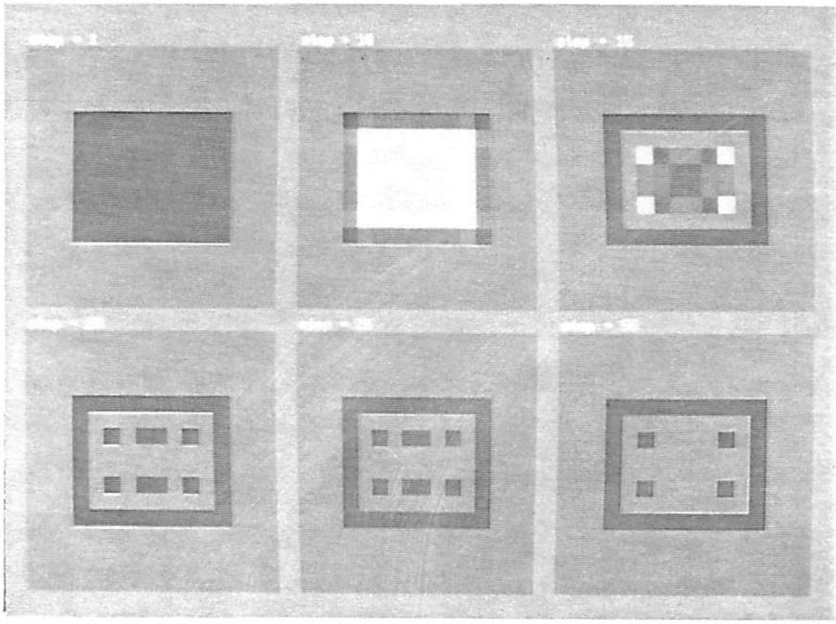


Fig. 3.23

0.0	0.0	0.0
0.0	2.0	0.0
0.0	0.0	0.0

(a)

-0.25	-0.25	-0.25
-0.25	2.0	-0.25
-0.25	-0.25	-0.25

(b)

Fig. 3.24

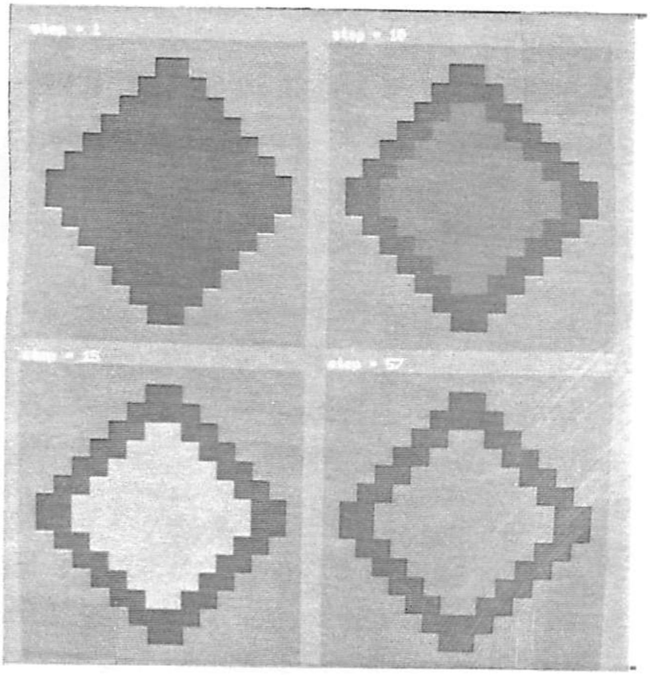


Fig. 3.25

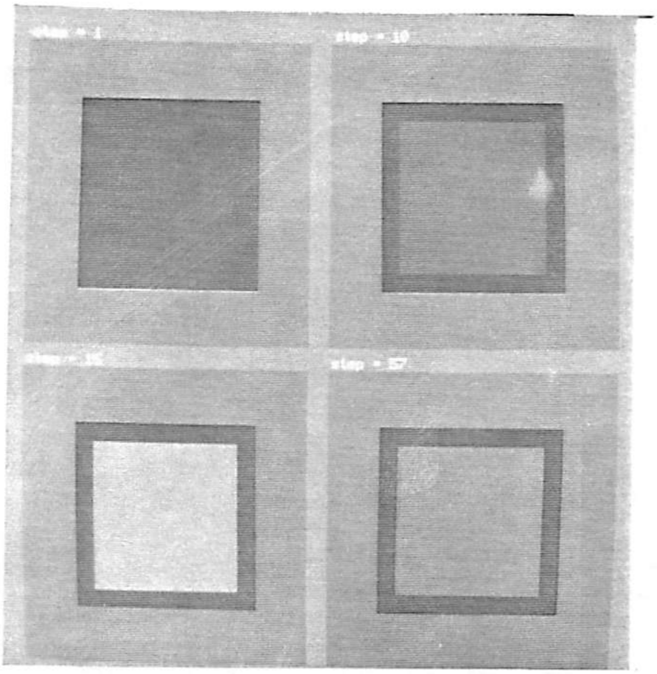


Fig. 3.26

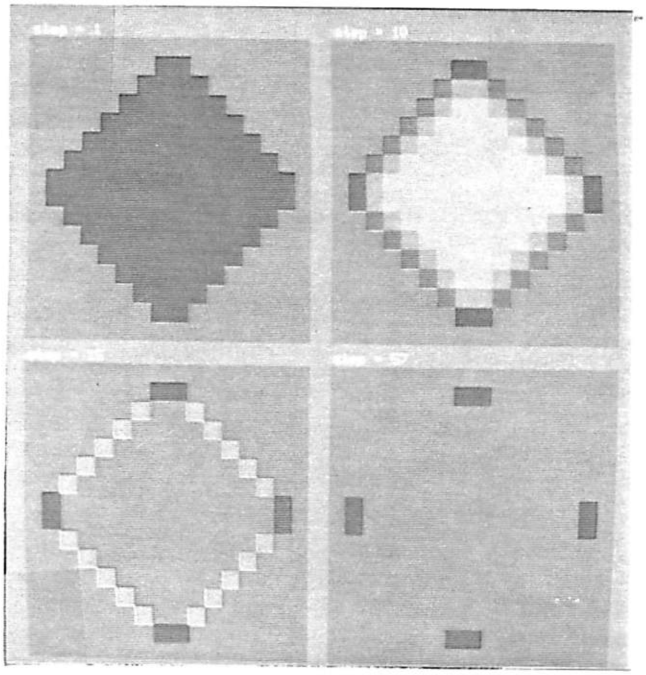


Fig. 3.27

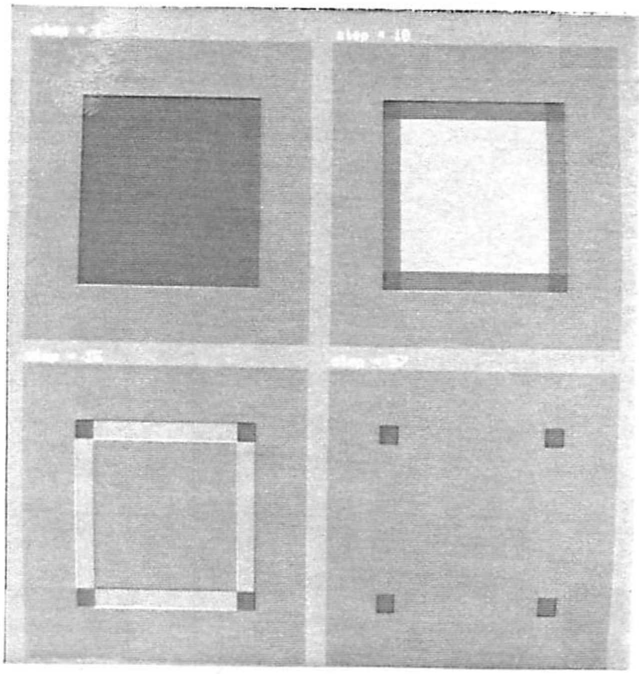


Fig. 3.28

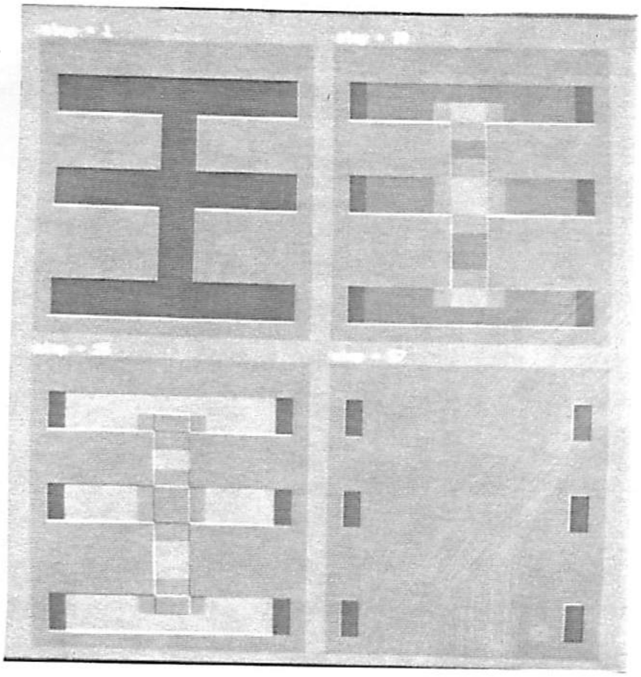


Fig. 3.29

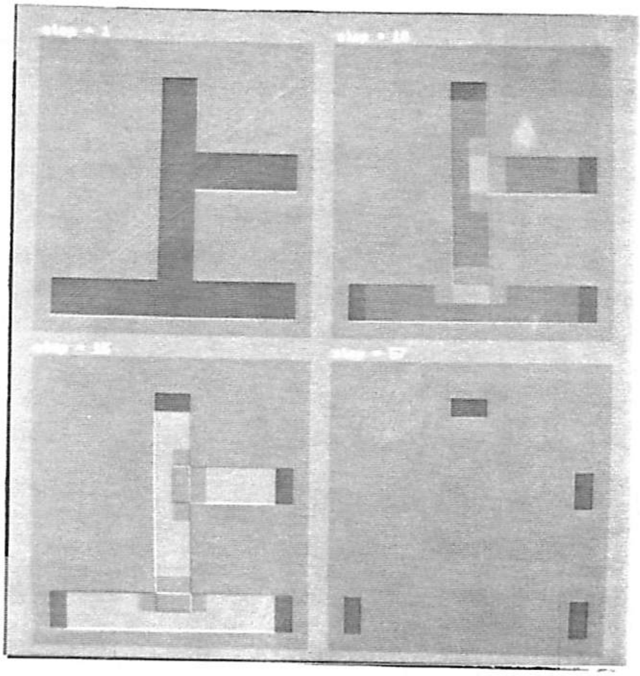


Fig. 3.30

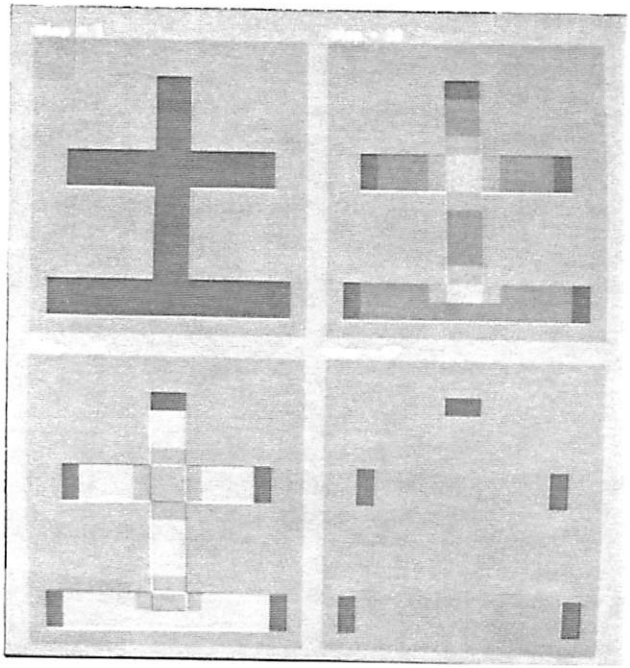


Fig. 3.31

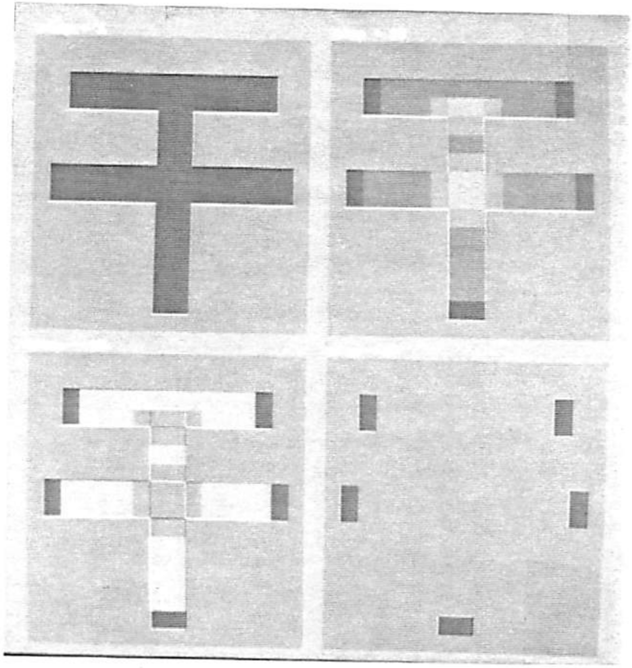


Fig. 3.32

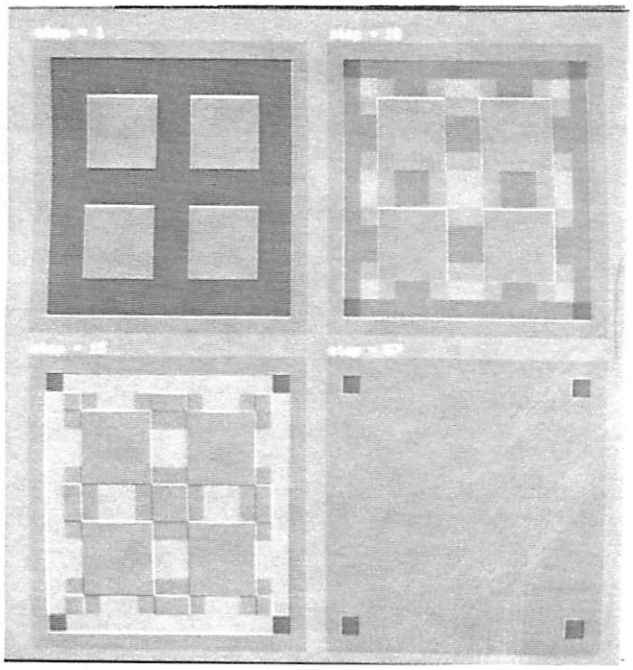


Fig. 3.33

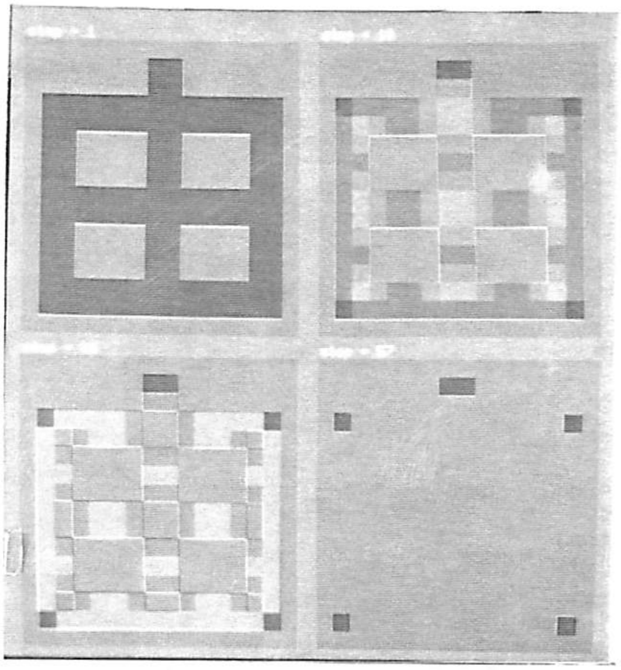


Fig. 3.34

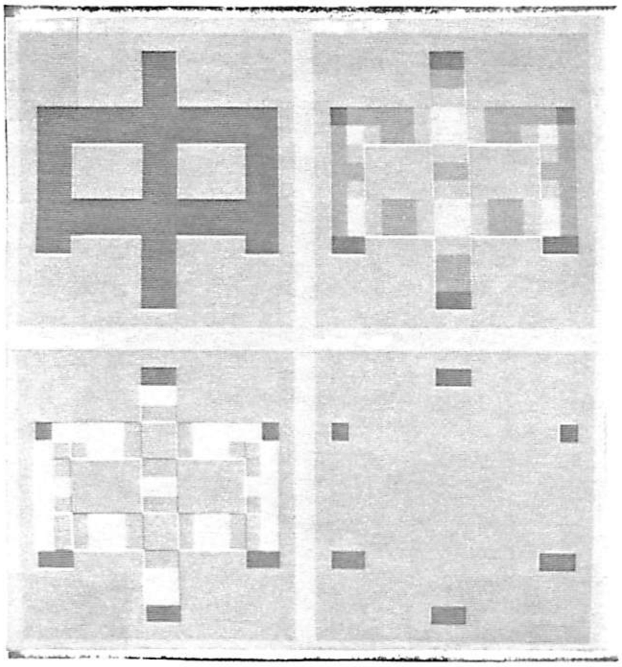


Fig. 3.35

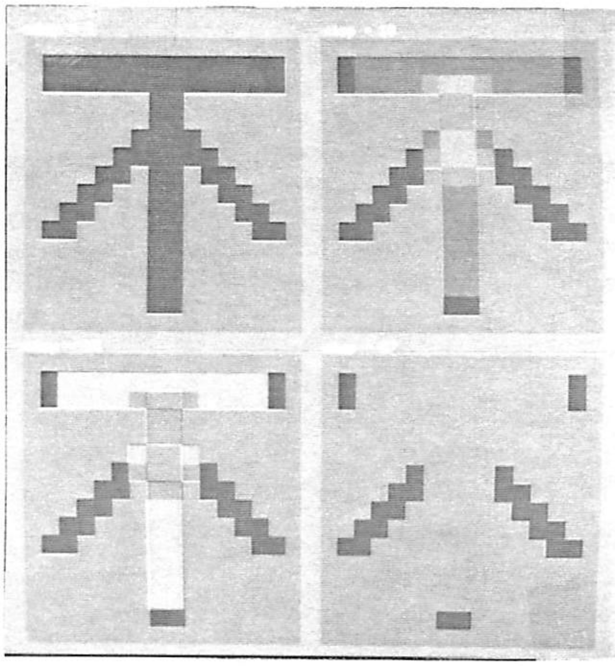


Fig. 3.36

APPENDIX

Table 3.1 Simple horizontal lines detector : Typical input file for circuit simulator PWLSPICE. The indices of nodes are coded as follow : the first number from the left stands for the type of the nodes in a cell, 1 for the state voltage node, 2 for the output voltage node; the second number from the left stands for the layer number; the third and fourth numbers stand for the rows and the fifth and sixth numbers stand for the columns, for instances, v(110102) means v_{x12}, and v(210304) means v_{y34}, and so on.

```
--------------------------------------------------------------------------------
        C110101            110101              0                1e-09
        R110101            110101              0                1000
        G1101010101   0  110101   210101   0             0.002
        G1101010102   0  110101   210102   0             0.001
        P210101            110101   210101     0                modpwl1
        R210101            210101              0                1
        C110102            110102                                -09
        R110102            11                                    00
        G1101020101   0  1101
        G1101020102   0  1101
        G1101020103   0  1101
        P210102            11                                   pwl1
        R210102            21
        C110103            11                                   -09
        R110103            11                                    0
        G1101030102   0  1101
        G1101030103   0  1101
        G1101030104   0  1101
        P210103            11                                   pwl1
        R210103            21
        C110104            11                                   09
        R110104            11                                   0
        G1101040103   0  1101
        G1101040104   0  1101
        P210104            11                                   pwl1
        R210104            21
        C110201            110201              0                1e-09
        R110201            110201              0                1000
        G1102010201   0  110201   210201   0             0.002
        G1102010202   0  110201   210202   0             0.001
        P210201            110201   210201     0                modpwl1
        R210201            210201              0                1
        C110202            110202              0                1e-09
        R110202            110202              0                1000
        G1102020201   0  110202   210201   0             0.001
--------------------------------------------------------------------------------
```

Please
Don't send this
Part of copies. We
have already in the
originals only we
need color copy
B/W master
copy

Table 3.1 continue

G1102020202	0	110202	210202	0		0.002	
G1102020203	0	110202	210203	0		0.001	
P210202		110202	210202		0		modpwl1
R210202		210202			0		1
C110203		110203			0		1e-09
R110203		110203			0		1000
G1102030202	0	110203	210202	0		0.001	
G1102030203	0	110203	210203	0		0.002	
G1102030204	0	110203	210204	0		0.001	
P210203		110203	210203		0		modpwl1
R210203		210203			0		1
C110204		110204			0		1e-09
R110204		110204			0		1000
G1102040203	0	110204	210203	0		0.001	
G1102040204	0	110204	210204	0		0.002	
P210204		110204	210204		0		modpwl1
R210204		210204			0		1
C110301		110301			0		1e-09
R110301		110301			0		1000
G1103010301	0	110301	210301	0		0.002	
G1103010302	0	110301	210302	0		0.001	
P210301		110301	210301		0		modpwl1
R210301		210301			0		1
C110302		110302			0		1e-09
R110302		110302			0		1000
G1103020301	0	110302	210301	0		0.001	
G1103020302	0	110302	210302	0		0.002	
G1103020303	0	110302	210303	0		0.001	
P210302		110302	210302		0		modpwl1
R210302		210302			0		1
C110303		110303			0		1e-09
R110303		110303			0		1000
G1103030302	0	110303	210302	0		0.001	
G1103030303	0	110303	210303	0		0.002	
G1103030304	0	110303	210304	0		0.001	
P210303		110303	210303		0		modpwl1
R210303		210303			0		1
C110304		110304			0		1e-09
R110304		110304			0		1000
G1103040303	0	110304	210303	0		0.001	
G1103040304	0	110304	210304	0		0.002	
P210304		110304	210304		0		modpwl1
R210304		210304			0		1
C110401		110401			0		1e-09
R110401		110401			0		1000
G1104010401	0	110401	210401	0		0.002	
G1104010402	0	110401	210402	0		0.001	
P210401		110401	210401		0		modpwl1
R210401		210401			0		1
C110402		110402			0		1e-09
R110402		110402			0		1000
G1104020401	0	110402	210401	0		0.001	
G1104020402	0	110402	210402	0		0.002	
G1104020403	0	110402	210403	0		0.001	
P210402		110402	210402		0		modpwl1
R210402		210402			0		1

Table 3.1 continue

```
C110403              110403              0                 1e-09
R110403              110403              0                 1000
G1104030402  0  110403  210402  0                 0.001
G1104030403  0  110403  210403  0                 0.002
G1104030404  0  110403  210404  0                 0.001
P210403              110403  210403      0                 modpwl1
R210403              210403              0                 1
C110404              110404              0                 1e-09
R110404              110404              0                 1000
G1104040403  0  110404  210403  0                 0.001
G1104040404  0  110404  210404  0                 0.002
P210404              110404  210404      0                 modpwl1
R210404              210404              0                 1
* This is a                PWL V              CCS model with a common node.
.model  modpwl1 pwl  term = 3 nseg = 3
+ ap= 0,0 bp= 0,0,0,0 cp= 0,0,-0.5,0.5 alphap= 1,0,1,0  betap=-1,1
.ic  v(110101)=-1  v(110102)=0.4  v(110103)=-0.8  v(110104)=-1
+    v(110201)=-0.4  v(110202)=-1  v(110203)=-0.8  v(110204)=-0.6
+    v(110301)=0.8  v(110302)=-0.4  v(110303)=0.8  v(110304)=1
+    v(110401)=-0.8  v(110402)=-0.6  v(110403)=-0.8  v(110404)=-1
.tran   0.1us    5us    UIC
.print tran   v(210101) v(210102) v(210103) v(210104)
.print tran   v(210201) v(210202) v(210203) v(210204)
.print tran   v(210301) v(210302) v(210303) v(210304)
.print tran   v(210401) v(210402) v(210403) v(210404)
.end
```

Table 3.2 Simple horizontal lines detector: Transient analysis.
(The indices of the circuit nodes are the same as those in Table 3.1)

Index	TIME	v(210101)	v(210102)	v(210103)	v(210104)
0	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
1	1.000000E-09	-1.000000E+00	3.985972E-01	-8.014028E-01	-1.000000E+00
2	2.000000E-09	-1.000000E+00	3.971916E-01	-8.028084E-01	-1.000000E+00
3	4.000000E-09	-1.000000E+00	3.943747E-01	-8.056253E-01	-1.000000E+00
4	8.000000E-09	-1.000000E+00	3.887070E-01	-8.112930E-01	-1.000000E+00
5	1.600000E-08	-1.000000E+00	3.772346E-01	-8.227654E-01	-1.000000E+00
6	3.200000E-08	-1.000000E+00	3.537300E-01	-8.462700E-01	-1.000000E+00
7	6.400000E-08	-1.000000E+00	3.043898E-01	-8.956102E-01	-1.000000E+00
8	1.280000E-07	-1.000000E+00	1.957343E-01	-1.000000E+00	-1.000000E+00
9	2.280000E-07	-1.000000E+00	5.811578E-03	-1.000000E+00	-1.000000E+00
10	3.280000E-07	-1.000000E+00	-2.041030E-01	-1.000000E+00	-1.000000E+00
11	4.280000E-07	-1.000000E+00	-4.361138E-01	-1.000000E+00	-1.000000E+00
12	5.280000E-07	-1.000000E+00	-6.925469E-01	-1.000000E+00	-1.000000E+00
13	6.280000E-07	-1.000000E+00	-9.759729E-01	-1.000000E+00	-1.000000E+00
14	7.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
15	8.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
16	9.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
17	1.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
18	1.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
19	1.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
20	1.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
21	1.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
22	1.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
23	1.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
24	1.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
25	1.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
26	1.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
27	2.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
28	2.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
29	2.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
30	2.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
31	2.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
32	2.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
33	2.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
34	2.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
35	2.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
36	2.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
37	3.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
38	3.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
39	3.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
40	3.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
41	3.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
42	3.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
43	3.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
44	3.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
45	3.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
46	3.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
47	4.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
48	4.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
49	4.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
50	4.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
51	4.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
52	4.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
53	4.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
54	4.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
55	4.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
56	4.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
57	5.000000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00

Table 3.2 continue

Index	TIME	v(210201)	v(210202)	v(210203)	v(210204)
0	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
1	1.000000E-09	-4.014014E-01	-1.000000E+00	-8.024038E-01	-6.014038E-01
2	2.000000E-09	-4.028042E-01	-1.000000E+00	-8.048114E-01	-6.028114E-01
3	4.000000E-09	-4.056126E-01	-1.000000E+00	-8.096343E-01	-6.056343E-01
4	8.000000E-09	-4.112463E-01	-1.000000E+00	-8.193262E-01	-6.113262E-01
5	1.600000E-08	-4.225817E-01	-1.000000E+00	-8.388959E-01	-6.228959E-01
6	3.200000E-08	-4.455265E-01	-1.000000E+00	-8.787950E-01	-6.467950E-01
7	6.400000E-08	-4.925355E-01	-1.000000E+00	-9.617566E-01	-6.977566E-01
8	1.280000E-07	-5.912155E-01	-1.000000E+00	-1.000000E+00	-8.087408E-01
9	2.280000E-07	-7.587119E-01	-1.000000E+00	-1.000000E+00	-9.991345E-01
10	3.280000E-07	-9.438395E-01	-1.000000E+00	-1.000000E+00	-1.000000E+00
11	4.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
12	5.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
13	6.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
14	7.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
15	8.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
16	9.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
17	1.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
18	1.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
19	1.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
20	1.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
21	1.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
22	1.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
23	1.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
24	1.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
25	1.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
26	1.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
27	2.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
28	2.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
29	2.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
30	2.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
31	2.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
32	2.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
33	2.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
34	2.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
35	2.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
36	2.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
37	3.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
38	3.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
39	3.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
40	3.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
41	3.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
42	3.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
43	3.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
44	3.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
45	3.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
46	3.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
47	4.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
48	4.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
49	4.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
50	4.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
51	4.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
52	4.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
53	4.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
54	4.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
55	4.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
56	4.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
57	5.000000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00

Table 3.2 continue

Index	TIME	v(210301)	v(210302)	v(210303)	v(210304)
0	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
1	1.000000E-09	8.004016E-01	-3.987970E-01	8.014026E-01	1.000000E+00
2	2.000000E-09	8.008048E-01	-3.975910E-01	8.028078E-01	1.000000E+00
3	4.000000E-09	8.016145E-01	-3.951729E-01	8.056235E-01	1.000000E+00
4	8.000000E-09	8.032533E-01	-3.903003E-01	8.112864E-01	1.000000E+00
5	1.600000E-08	8.066099E-01	-3.804076E-01	8.227396E-01	1.000000E+00
6	3.200000E-08	8.136486E-01	-3.600176E-01	8.461675E-01	1.000000E+00
7	6.400000E-08	8.291051E-01	-3.166991E-01	8.952019E-01	1.000000E+00
8	1.280000E-07	8.662148E-01	-2.189430E-01	1.000000E+00	1.000000E+00
9	2.280000E-07	9.436895E-01	-4.146832E-02	1.000000E+00	1.000000E+00
10	3.280000E-07	1.000000E+00	1.617292E-01	1.000000E+00	1.000000E+00
11	4.280000E-07	1.000000E+00	3.892796E-01	1.000000E+00	1.000000E+00
12	5.280000E-07	1.000000E+00	6.407828E-01	1.000000E+00	1.000000E+00
13	6.280000E-07	1.000000E+00	9.187599E-01	1.000000E+00	1.000000E+00
14	7.280000E-07	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
15	8.280000E-07	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
16	9.280000E-07	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
17	1.028000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
18	1.128000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
19	1.228000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
20	1.328000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
21	1.428000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
22	1.528000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
23	1.628000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
24	1.728000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
25	1.828000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
26	1.928000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
27	2.028000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
28	2.128000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
29	2.228000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
30	2.328000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
31	2.428000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
32	2.528000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
33	2.628000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
34	2.728000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
35	2.828000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
36	2.928000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
37	3.028000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
38	3.128000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
39	3.228000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
40	3.328000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
41	3.428000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
42	3.528000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
43	3.628000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
44	3.728000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
45	3.828000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
46	3.928000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
47	4.028000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
48	4.128000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
49	4.228000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
50	4.328000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
51	4.428000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
52	4.528000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
53	4.628000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
54	4.728000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
55	4.828000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
56	4.928000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00
57	5.000000E-06	1.000000E+00	1.000000E+00	1.000000E+00	1.000000E+00

Table 3.2 continue

Index	TIME	v(210401)	v(210402)	v(210403)	v(210404)
0	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00	0.000000E+00
1	1.000000E-09	-8.014036E-01	-6.022060E-01	-8.024046E-01	0.000000E+00
2	2.000000E-09	-8.028108E-01	-6.044181E-01	-8.048138E-01	-1.000000E+00
3	4.000000E-09	-8.056326E-01	-6.088542E-01	-8.096416E-01	-1.000000E+00
4	8.000000E-09	-8.113198E-01	-6.177994E-01	-8.193529E-01	-1.000000E+00
5	1.600000E-08	-8.228717E-01	-6.359847E-01	-8.390014E-01	-1.000000E+00
6	3.200000E-08	-8.467047E-01	-6.735635E-01	-8.792236E-01	-1.000000E+00
7	6.400000E-08	-8.974488E-01	-7.537920E-01	-9.635456E-01	-1.000000E+00
8	1.280000E-07	-1.000000E+00	-9.312657E-01	-1.000000E+00	-1.000000E+00
9	2.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
10	3.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
11	4.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
12	5.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
13	6.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
14	7.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
15	8.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
16	9.280000E-07	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
17	1.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
18	1.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
19	1.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
20	1.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
21	1.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
22	1.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
23	1.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
24	1.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
25	1.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
26	1.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
27	2.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
28	2.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
29	2.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
30	2.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
31	2.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
32	2.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
33	2.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
34	2.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
35	2.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
36	2.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
37	3.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
38	3.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
39	3.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
40	3.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
41	3.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
42	3.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
43	3.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
44	3.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
45	3.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
46	3.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
47	4.028000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
48	4.128000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
49	4.228000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
50	4.328000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
51	4.428000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
52	4.528000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
53	4.628000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
54	4.728000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
55	4.828000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
56	4.928000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00
57	5.000000E-06	-1.000000E+00	-1.000000E+00	-1.000000E+00	-1.000000E+00

FIGURE CAPTIONS:

Fig. 2.1: A two-dimensional cellular neural network. The circuit size is 4×4. The squares are the circuit units called *cells*. The links between the *cells* indicate that there are interactions between the linked *cells*.

Fig. 2.2: The neighborhood of cell $C(i,j)$ defined by (2.1) for $r = 1$, $r = 2$ and $r = 3$ respectively.

Fig. 2.3: An example of a *cell* circuit. C is a linear capacitor; R_x, R_u and R_y are linear resistors; I is an independent voltage source; $I_{xu}(i,j;k,l)$ and $I_{xy}(i,j;k,l)$ are linear voltage controlled current sources with the characteristics $I_{xy}(i,j;k,l) = A(i,j;k,l)v_{ykl}$ and $I_{xu}(i,j;k,l) = B(i,j;k,l)v_{ukl}$ for all $C(i, j) \in N(i, j)$; $I_{yx} = \frac{1}{R_y} f(v_{xij})$ is a piecewise-linear voltage controlled current source with its characteristic $f(\cdot)$ as shown in Fig. 2.4; E_{ij} is an independent voltage source.

Fig. 2.4: The characteristic of the nonlinear controlled source.

Fig. 2.5: The characteristic of the nonlinear resistor in the equivalent *cell* circuit.

Fig. 2.6: The steady state equivalent circuit of a *cell* in cellular neural networks.

Fig. 2.7: *Dynamic routes* and *equilibrium points* of the equivalent circuit for different values of $g(t)$.

Fig. 3.1: Input and output images for the simple example. (a) The input image to be processed; (b) the output image of the horizontal line detector; (c) the output image of the vertical line detector.

Fig. 3.2: A typical templet of an interactive cell operator. The unit used here is $10^{-3}\Omega^{-1}$.

Fig. 3.3: The cellular neural network description file of the simple horizontal line detector for the circuit simulation preprocessor CELL.

Fig. 3.4: The data file of CELL for the image in Fig. 3.1a.

Fig. 3.5: Templets for various interactive cell operators for noise removing cellular neural networks. The unit used here is $10^{-3}\Omega^{-1}$.

Fig. 3.6: The simulation result of a noise removing cellular neural network. Here $\sigma = 0.2$ and the interactive cell operator is defined by the templet in Fig. 3.5(a). (a) the upper left picture, is the input image; (b) the upper right picture, is the image at time step 10; (c) the lower left picture, is the image at time step 20; (d) the lower right picture, is the image at time step 30 (output image). (The color in this picture and all the following pictures is designed to stand for the gray levels of the pixels in the pictures, where

background = light blue;

$[-1.0, -\frac{7}{8}]$ = greenish blue; $(-\frac{7}{8}, -\frac{6}{8}\ ]$ = blue green;

$(-\frac{6}{8}, -\frac{5}{8}\ ]$ = bluish green; $(-\frac{5}{8}, -\frac{4}{8}\ ]$ = green;

$(-\frac{4}{8}, -\frac{3}{8}\ ]$ = yellowish green; $(-\frac{3}{8}, -\frac{2}{8}\ ]$ = green yellow;

$(-\frac{2}{8}, -\frac{1}{8}\ ]$ = greenish yellow; $(-\frac{1}{8}, 0.0\ ]$ = yellow;

$(0.0, -\frac{1}{8}\ ]$ = orangish yellow; $(\frac{1}{8}, \frac{2}{8}\ ]$ = yellow orange;

$(\frac{2}{8}, \frac{3}{8}]$ = yellowish orange; $(\frac{3}{8}, \frac{4}{8}]$ = orange;

$(\frac{4}{8}, \frac{5}{8}]$ = reddish orange; $(\frac{5}{8}, \frac{6}{8}]$ = orange red;

$(\frac{6}{8}, \frac{7}{8}]$ = orangish red; $(\frac{7}{8}, 1.0]$ = red.)

Fig. 3.7: Simulation results of a noise removing cellular neural network. Here $\sigma = 0.4$ and the interactive cell operator is defined by the templet in Fig. 3.5(a). (a) the upper left picture, is the input image; (b) the upper right picture, is the image at time step 10; (c) the lower left picture, is the image at time step 20; (d) the lower right picture, is the image at time step 30 (output image).

Fig. 3.8: Simulation results of a noise removing cellular neural network. Here $\sigma = 0.2$ and the interactive cell operator is defined by the templet in Fig. 3.5(b). (a) the upper left picture, is the input image; (b) the upper right picture, is the image at time step 10; (c) the lower left picture, is the image at time step 20; (d) the lower right picture, is the image at time step 30 (output image).

Fig. 3.9: Simulation results of a noise removing cellular neural network. Here $\sigma = 0.4$ and the interactive cell operator is defined by the templet in Fig. 3.5(b). (a) the upper left picture, is the input image; (b) the upper right picture, is the image at time step 10; (c) the lower left picture, is the image at time step 20; (d) the lower right picture, is the image at time step 30 (output image).

Fig. 3.10: Simulation results of a noise removing cellular neural network. Here $\sigma = 0.2$ and the interactive cell operator is defined by the templet in Fig. 3.5(c). (a) the upper left picture, is the input image; (b) the upper middle picture, is the image at time step 10; (c) the upper right picture, is the image at time step 20; (d) the lower left picture, is the image at time

step 30; (e) the lower middle picture, is the image at time step 40; (f) the lower right picture, is the image at time step 57 (output image).

Fig. 3.11: Simulation results of a noise removing cellular neural network. Here $\sigma = 0.4$ and the interactive cell operator is defined by the templet in Fig. 3.5(c). (a) the upper left picture, is the input image; (b) the upper middle picture, is the image at time step 10; (c) the upper right picture, is the image at time step 20; (d) the lower left picture, is the image at time step 30; (e) the lower middle picture, is the image at time step 40; (f) the lower right picture, is the image at time step 58 (output image).

Fig. 3.12: Simulation results of a noise removing cellular neural network. Here the interactive cell operator is defined by the templet in Fig. 3.5(a). (a) the upper left picture, is the input image; (b) the upper middle picture, is the image at time step 10; (c) the upper right picture, is the image at time step 20; (d) the lower left picture, is the image at time step 30; (e) the lower middle picture, is the image at time step 40; (f) the lower right picture, is the image at time step 57 (output image).

Fig. 3.13: Simulation results of a noise removing cellular neural network. Here the interactive cell operator is defined by the templet in Fig. 3.5(d). (a) the upper left picture, is the input image; (b) the upper middle picture, is the image at time step 10; (c) the upper right picture, is the image at time step 20; (d) the lower left picture, is the image at time step 30; (e) the lower middle picture, is the image at time step 40; (f) the lower right picture, is the image at time step 57 (output image).

Fig. 3.14: Simulation results of a noise removing cellular neural network. Here $\sigma = 0.6$ and the interactive cell operator is defined by the templet in Fig. 3.5(b). (a) the upper left picture, is the input image; (b) the upper right picture, is the image at time step 10; (c) the lower left picture, is the image at time step 20; (d) the lower right picture, is the image at time

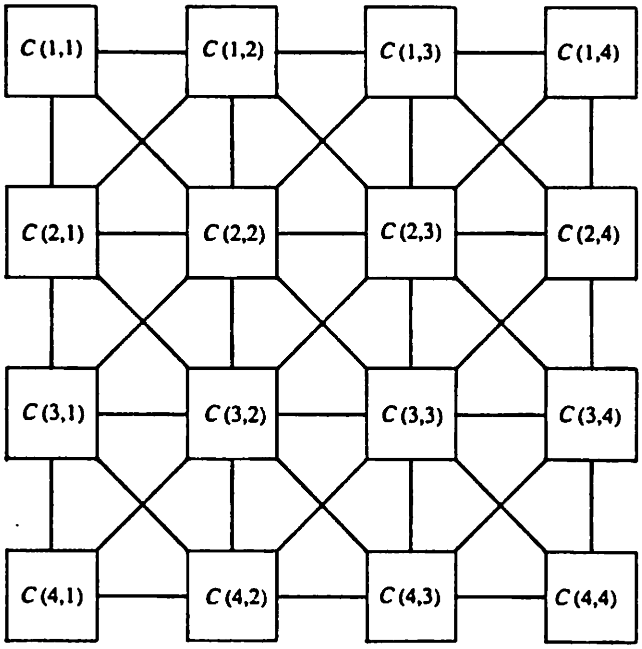
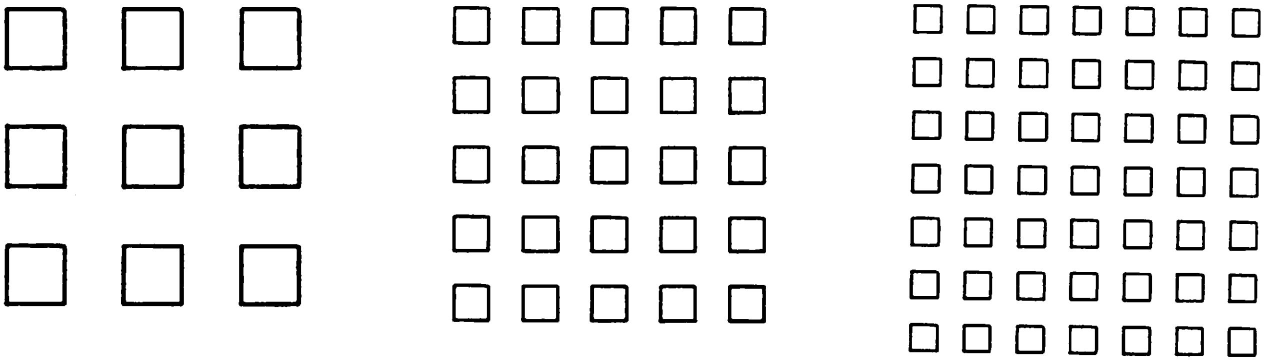
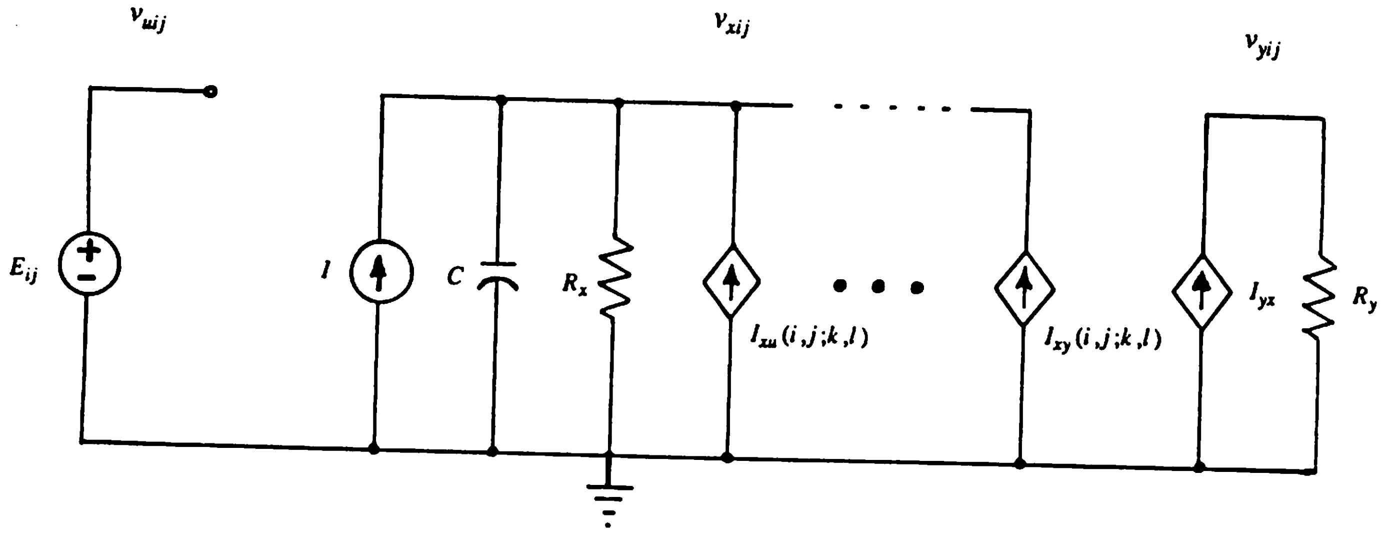


Fig. 2.1



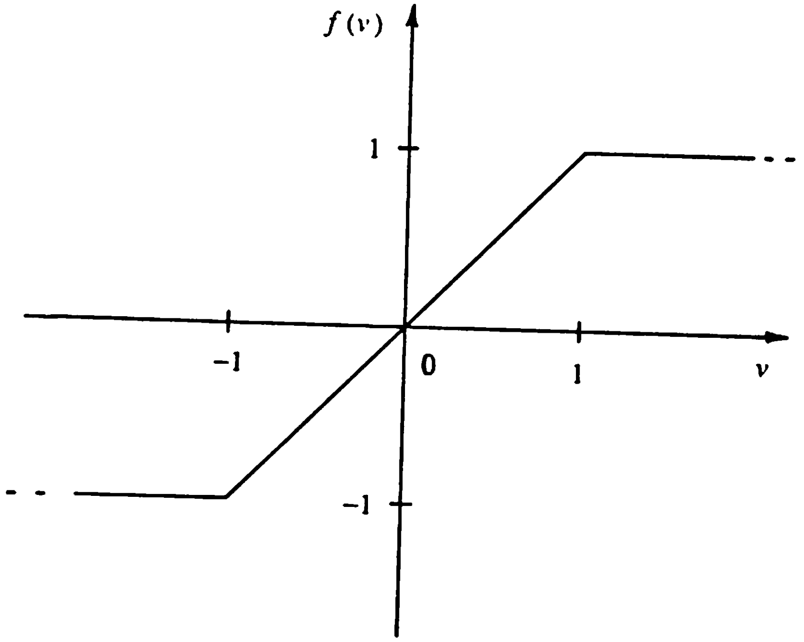
$r = 1$ $r = 2$ $r = 3$

Fig. 2.2



$I_{xu}(i,j;k,l) = B(i,j;k,l)v_{ukl}$, $I_{xy}(i,j;k,l) = A(i,j;k,l)v_{ykl}$, $I_{yx} = 0.5\left[\,|v_{xij}+1| - |v_{xij}-1|\,\right]$.

Fig. 2.3



$$f(v) = \frac{1}{2}\left[\,|\,v+1\,| - |\,v-1\,|\,\right]$$

Fig. 2.4

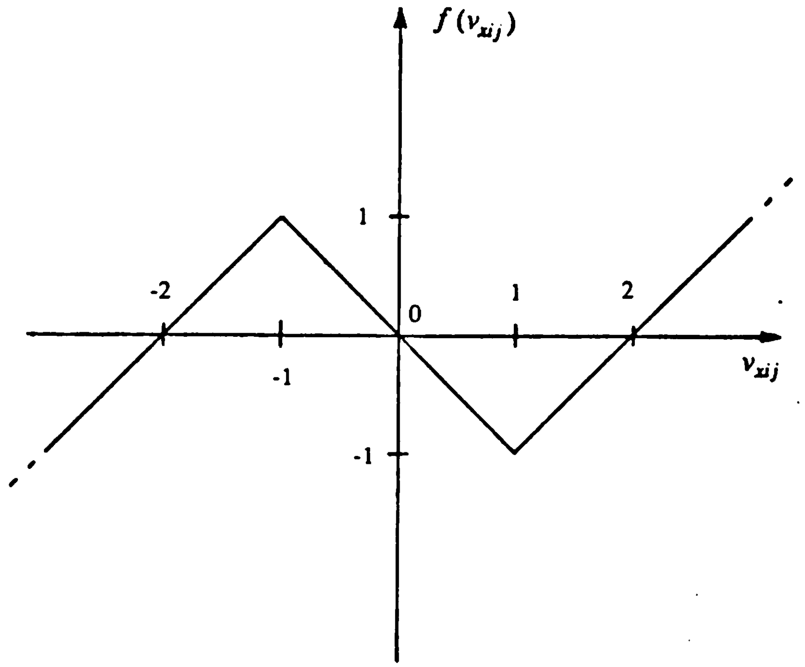


Fig. 2.5

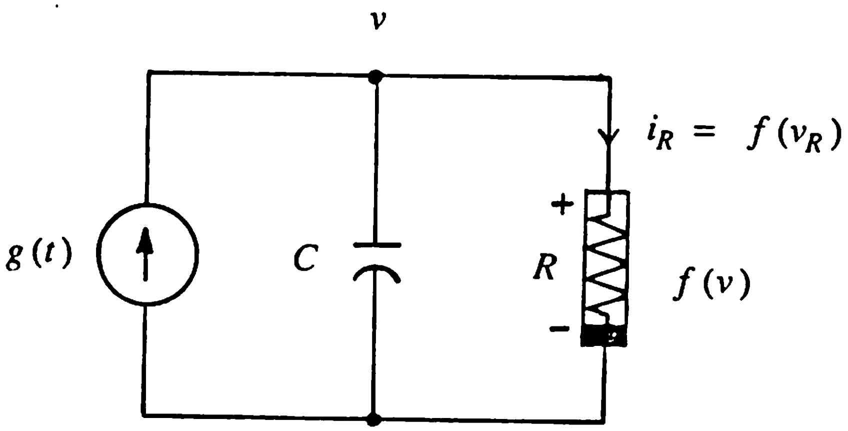
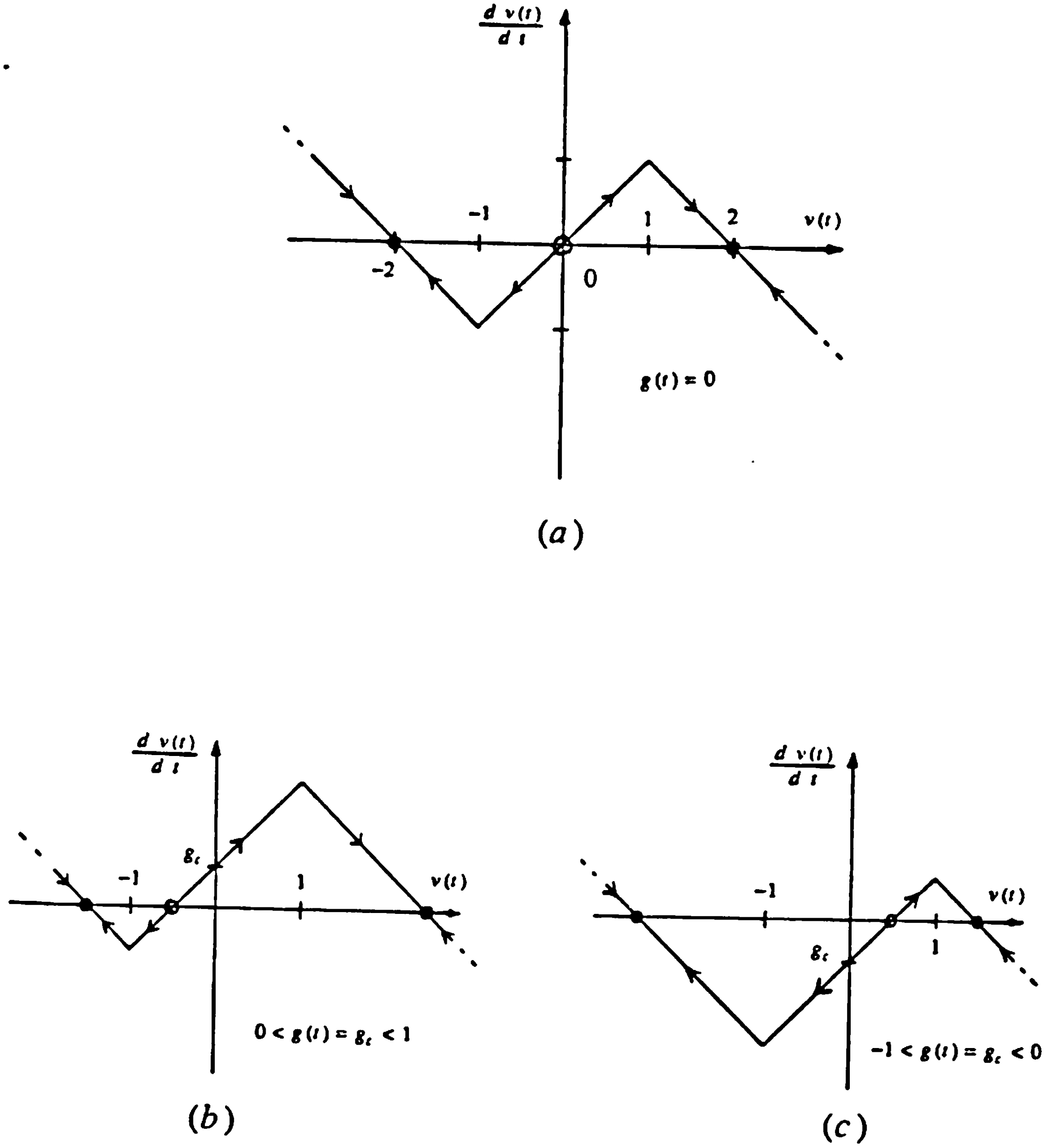


Fig. 2.6

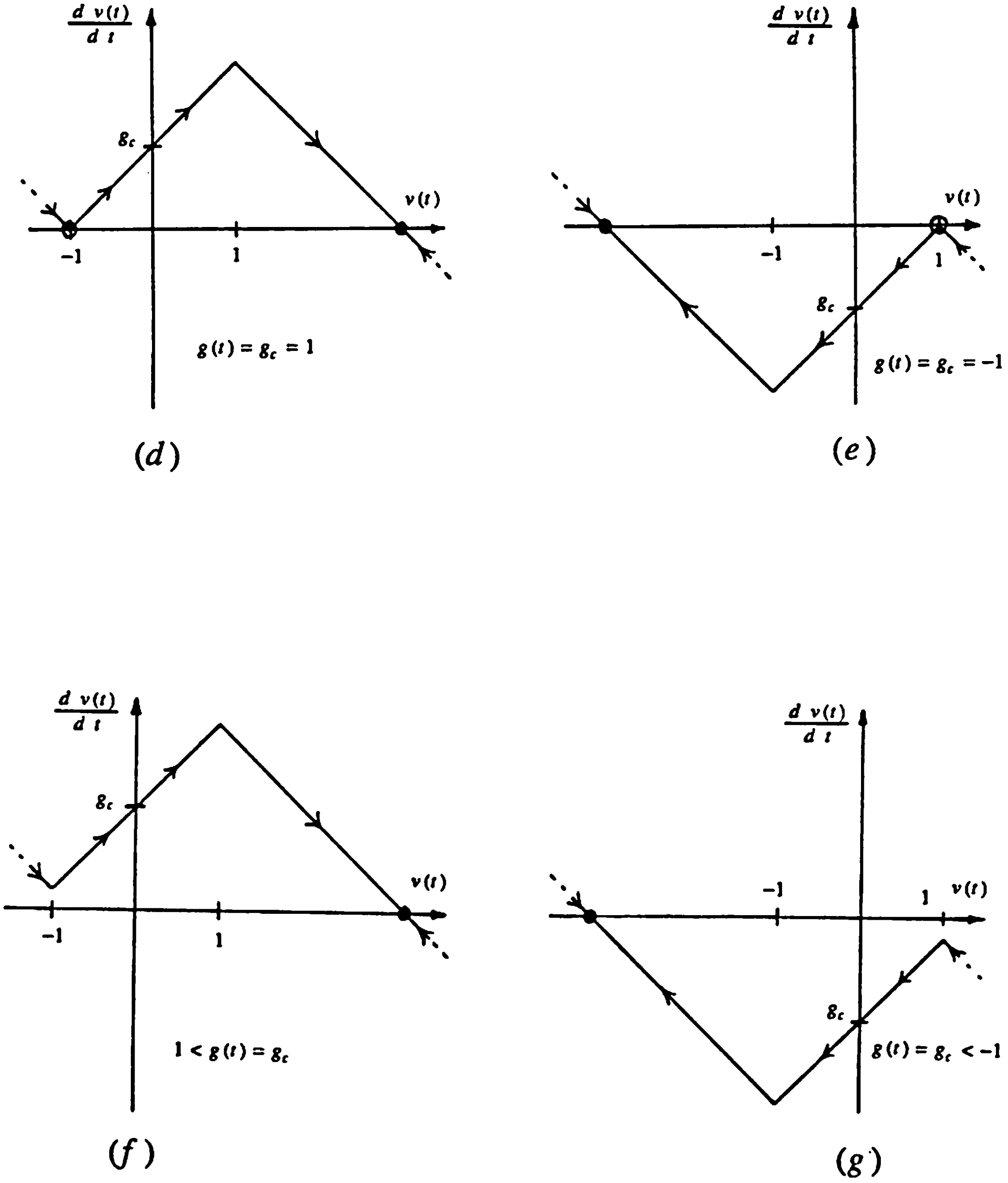


(a)

(b)

(c)

Fig. 2.7 (a, b, c)



$\frac{d\ v(t)}{d\ t}$
g_c
$v(t)$
-1
1
$g(t) = g_c = 1$
(d)
$\frac{d\ v(t)}{d\ t}$
$v(t)$
-1
1
g_c
$g(t) = g_c = -1$
(e)
$\frac{d\ v(t)}{d\ t}$
g_c
$v(t)$
-1
1
$1 < g(t) = g_c$
(f)
$\frac{d\ v(t)}{d\ t}$
-1
1
$v(t)$
g_c
$g(t) = g_c < -1$
(g)

Fig. 2.7 (d, e, f, g)

-1.0	0.4	-0.8	-1.0
-0.4	-1.0	-0.8	-0.6
0.8	-0.4	0.8	1.0
-0.8	-0.6	-0.8	-1.0

(a)

-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
1.0	1.0	1.0	1.0
-1.0	-1.0	-1.0	-1.0

(b)

-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0
-1.0	-1.0	-1.0	-1.0

(c)

Fig. 3.1

0.0	0.0	0.0
1.0	2.0	1.0
0.0	0.0	0.0

Fig. 3.2

```
**************************************************************************

name_of_circuit  "simple horizontal lines detector"
comments  "An example of the application of analog cellular circuits."
circuit_size  4 4
vector_dimension  1
c1  1.0E-9
r1  1000
y1  "pwl  term = 3 nseg = 3"
    "ap= 0,0 bp= 0,0,0,0 cp= 0,0,-0.5,0.5 alphap= 1,0,1,0  betap=-1,1"
a11 size 3
    templet 0.0    0.0    0.0
            0.001  0.002  0.001
            0.0    0.0    0.0
data_type x1
transient "0.1us    5us    UIC"
output    y1

**************************************************************************
```

Fig. 3.3

```
**************************************************************************

       This is the initial state of the horizontal line detector.

       Where      "." : -1.0;
                  "1" : -0.8;
                  "2" : -0.6;
                  "3" : -0.4;
                  "4" : -0.2;
                  "5" :  0.0;
                  "6" :  0.2;
                  "7" :  0.4;
                  "8" :  0.6;
                  "9" :  0.8;
                  "*" :  1.0.
$
                              .71.
                              3.12
                              939*
                              121.

**************************************************************************
```

Fig. 3.4

0.0	1.0	0.0
1.0	2.0	1.0
0.0	1.0	0.0

(a)

0.0	1.0	0.0
1.0	4.0	1.0
0.0	1.0	0.0

(b)

0.5	1.0	0.5
1.0	4.0	1.0
0.5	1.0	0.5

(c)

0.0	0.0	0.0
0.0	4.0	0.0
0.0	0.0	0.0

(d)

Fig. 3.5

0.0	1.0	0.0
1.0	2.0	1.0
0.0	1.0	0.0

(a)

0.0	1.0	0.0
1.0	4.0	1.0
0.0	1.0	0.0

(b)

0.5	1.0	0.5
1.0	4.0	1.0
0.5	1.0	0.5

(c)

0.0	0.0	0.0
0.0	4.0	0.0
0.0	0.0	0.0

(d)

Fig. 3.5

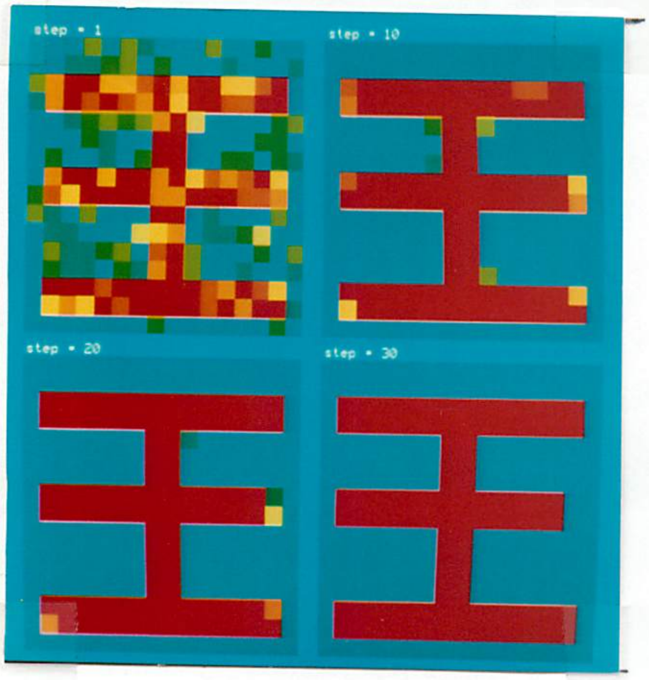


Fig. 3.6

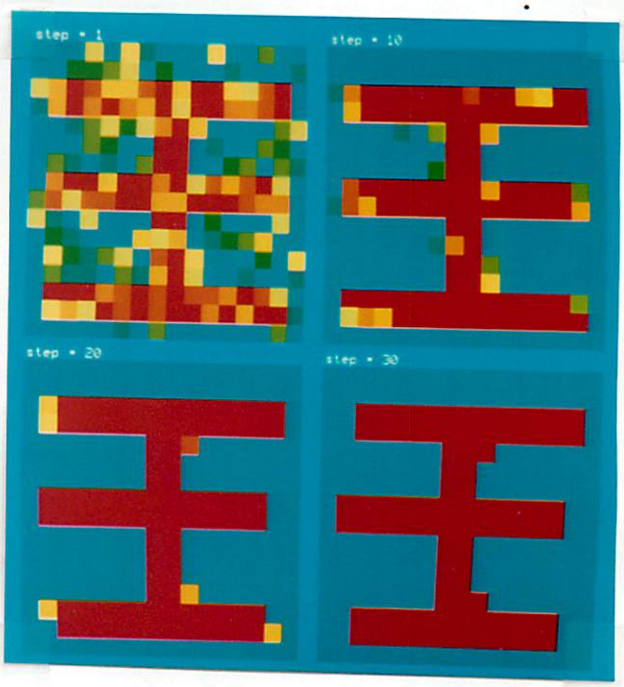


Fig. 3.7

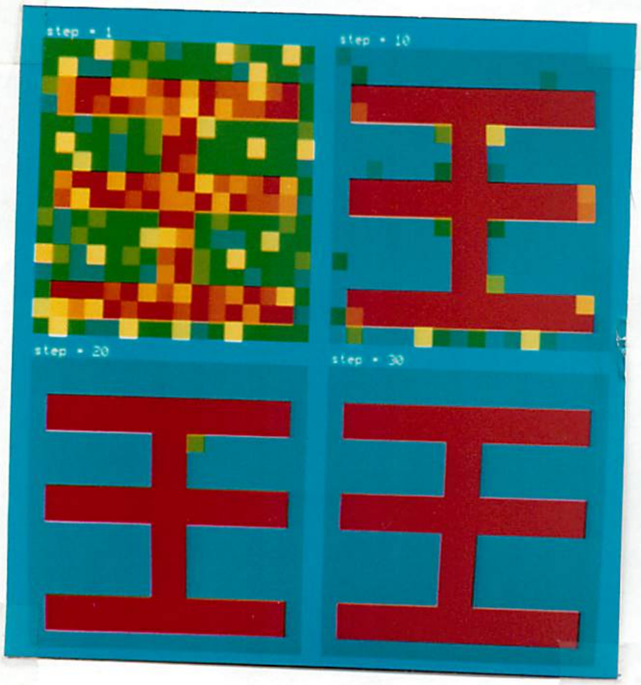


Fig. 3.8

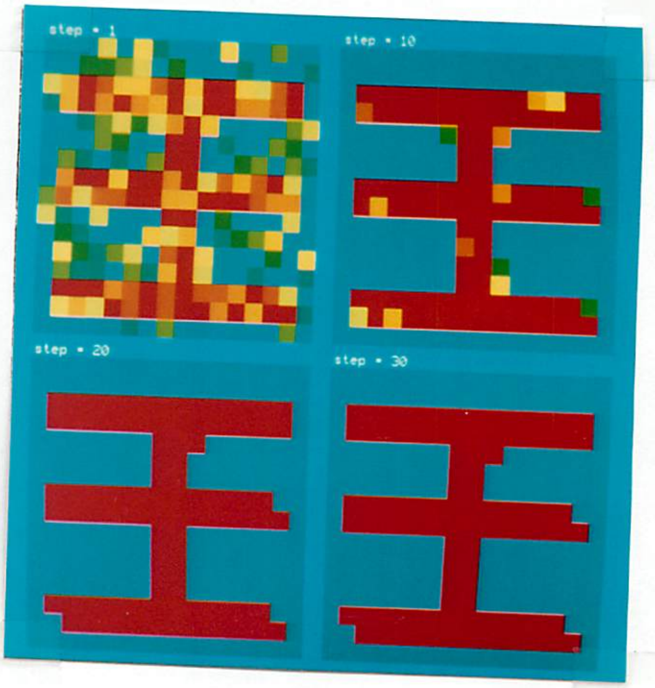


Fig. 3.9

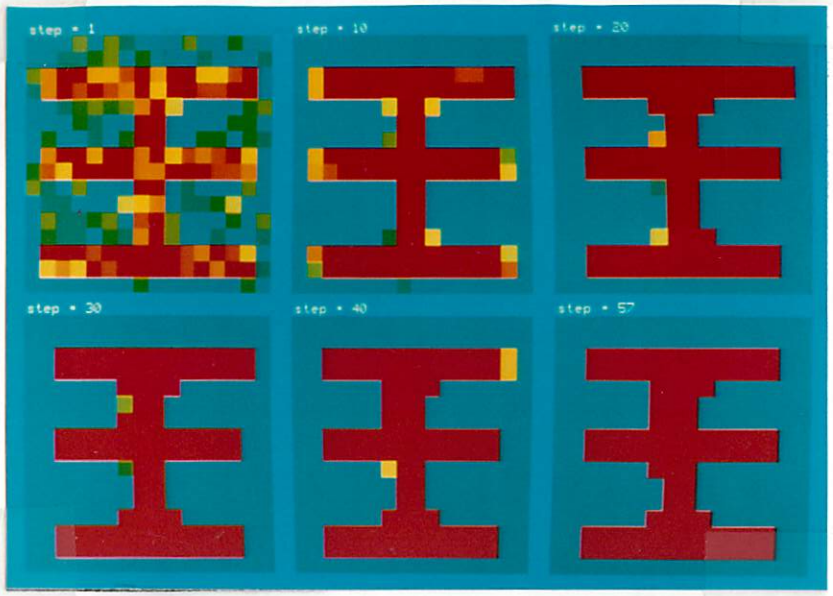


Fig. 3.10

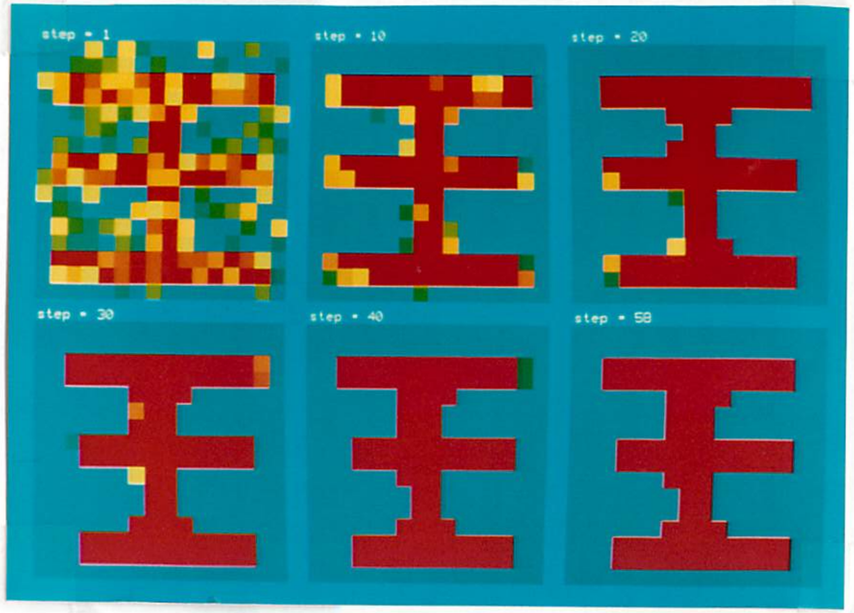


Fig. 3.11

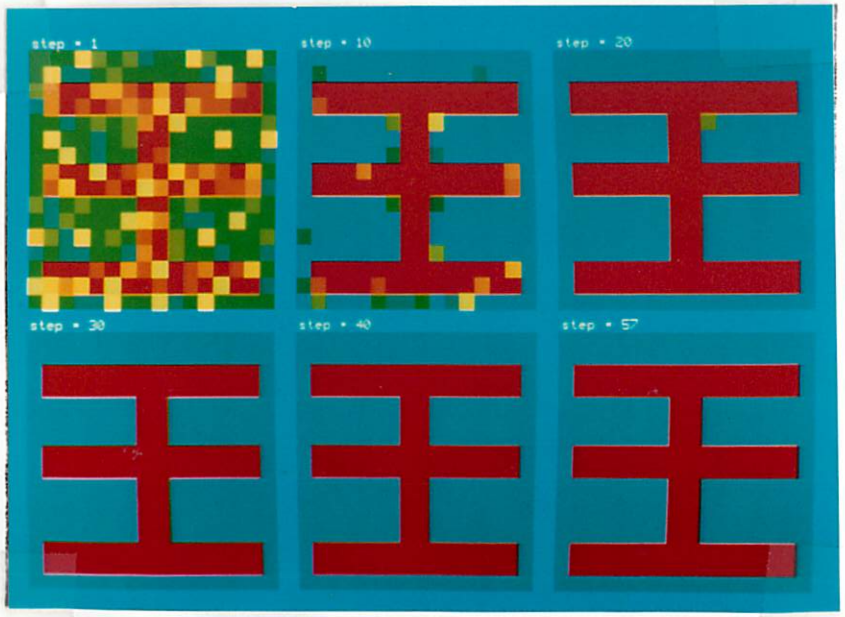


Fig. 3.12

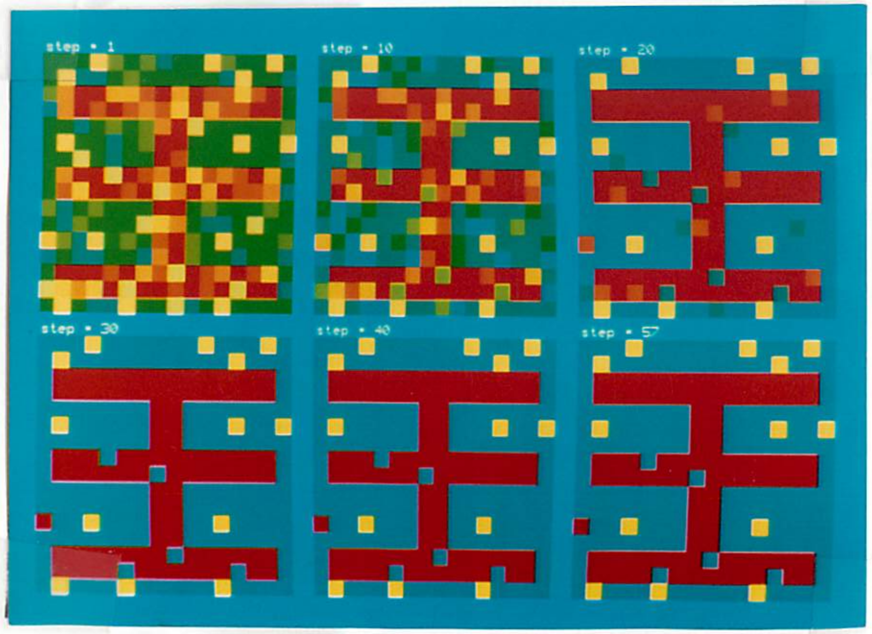


Fig. 3.13

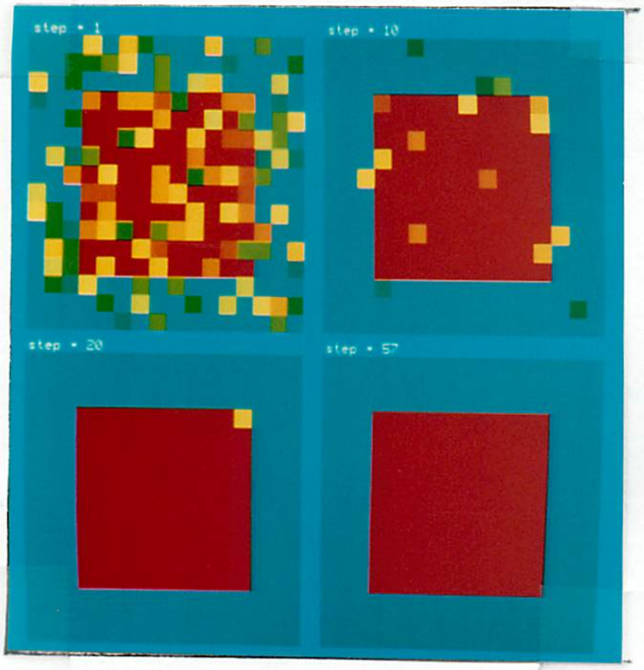


Fig. 3.14

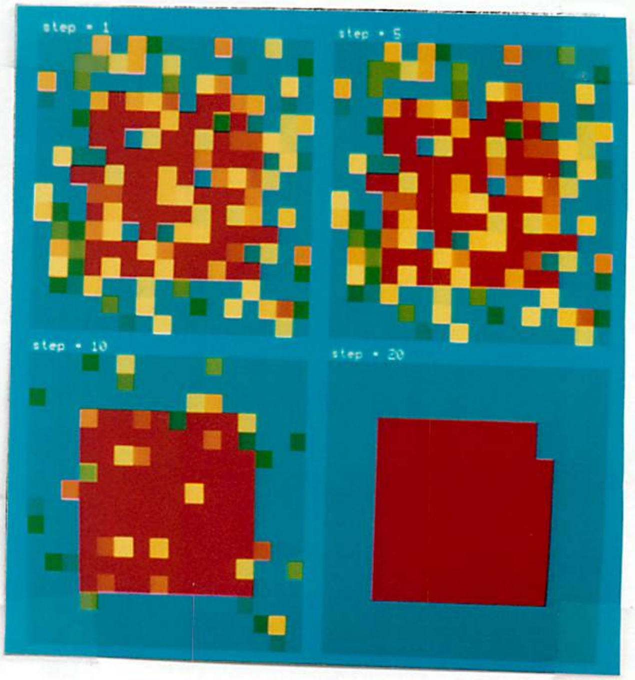


Fig. 3.15

0.0	-0.5	0.0
-0.5	2.0	-0.5
0.0	-0.5	0.0

Fig. 3.16

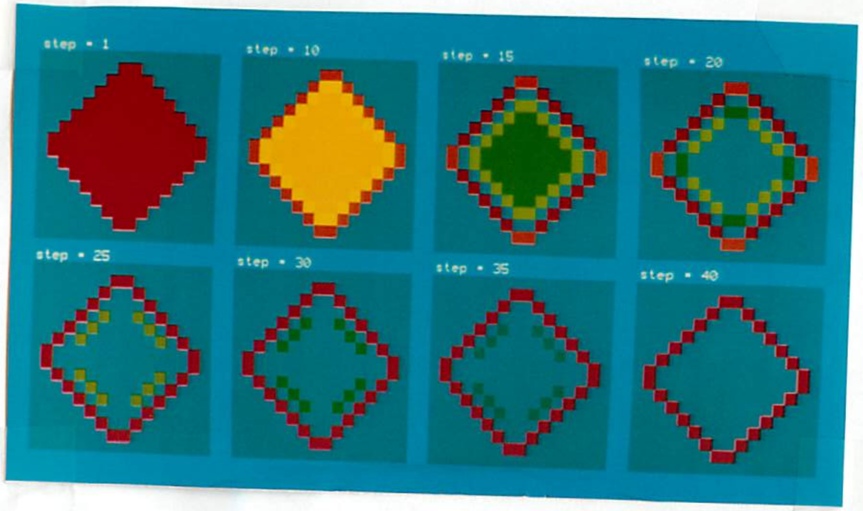


Fig. 3.17

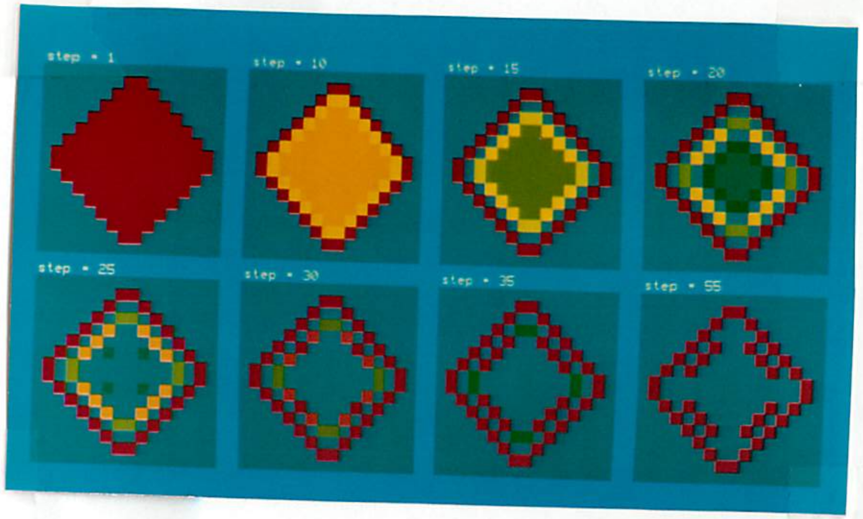


Fig. 3.18

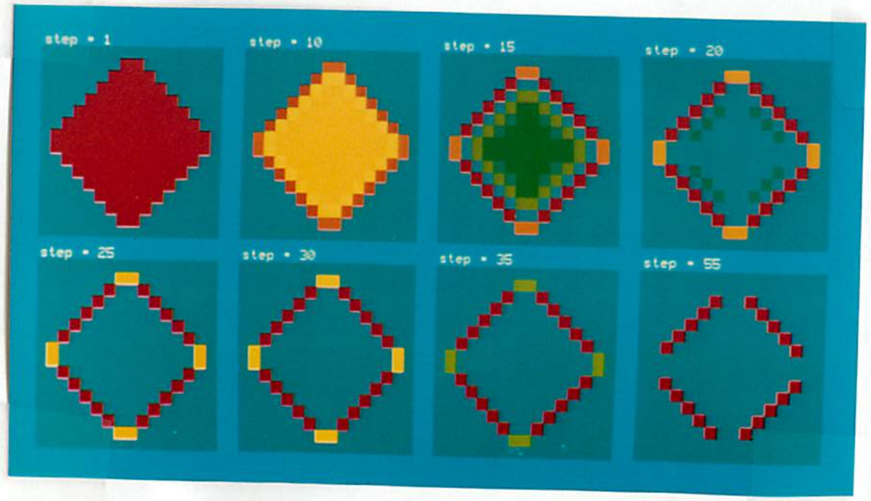


Fig. 3.19

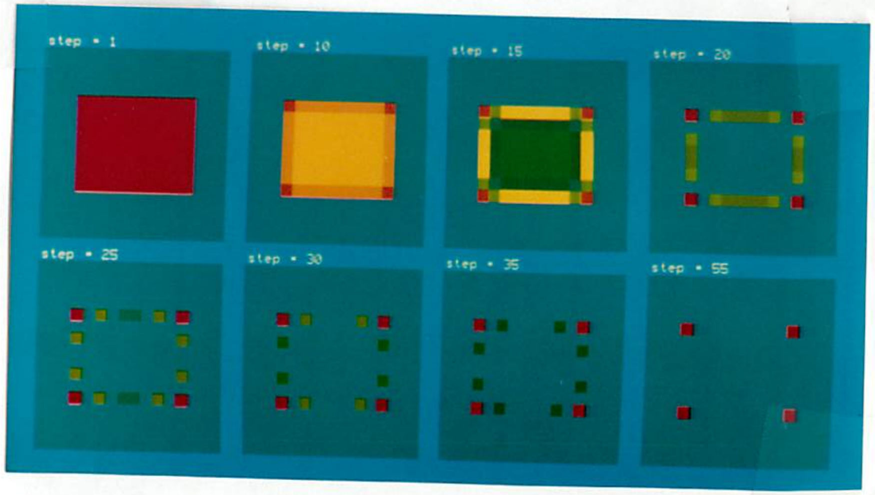


Fig. 3.20

-1	-1	-1
-1	-1	-1
-1	-1	-1

(a)

1	1	1
1	1	1
1	1	1

(b)

-1	-1	-1
-1	-1	-1
1	1	1

(c)

-1	-1	-1
1	1	1
1	1	1

(d)

-1	-1	-1
-1	-1	-1
-1	-1	1

(e)

-1	-1	-1
-1	1	1
-1	1	1

(f)

Fig. 3.21

0.0	-1.0	0.0
-1.0	4.0	-1.0
0.0	-1.0	0.0

Fig. 3.22

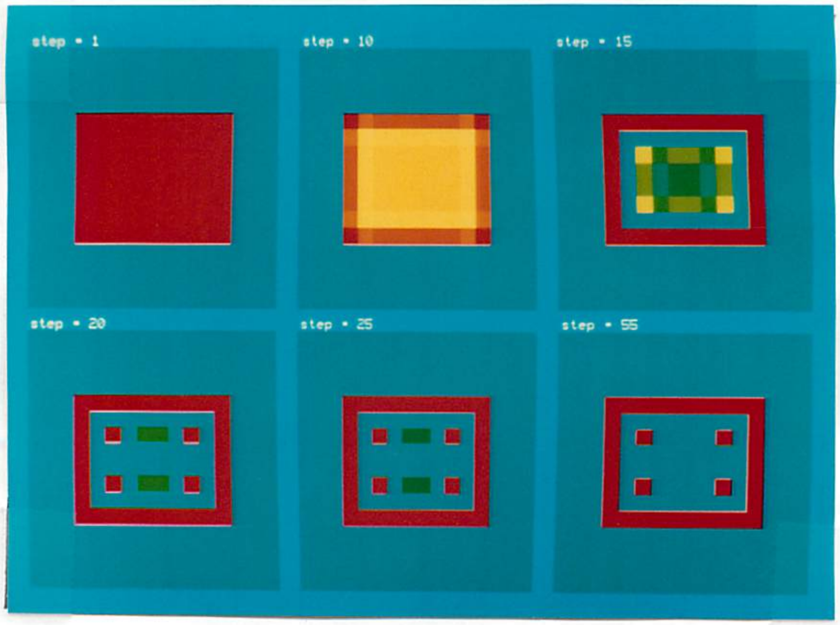


Fig. 3.23

0.0	0.0	0.0
0.0	2.0	0.0
0.0	0.0	0.0

(a)

-0.25	-0.25	-0.25
-0.25	2.0	-0.25
-0.25	-0.25	-0.25

(b)

Fig. 3.24

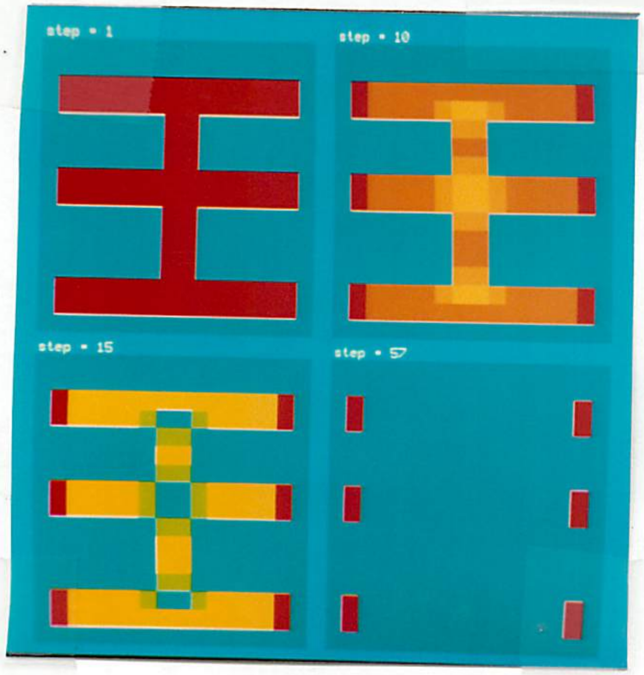


Fig. 3.29

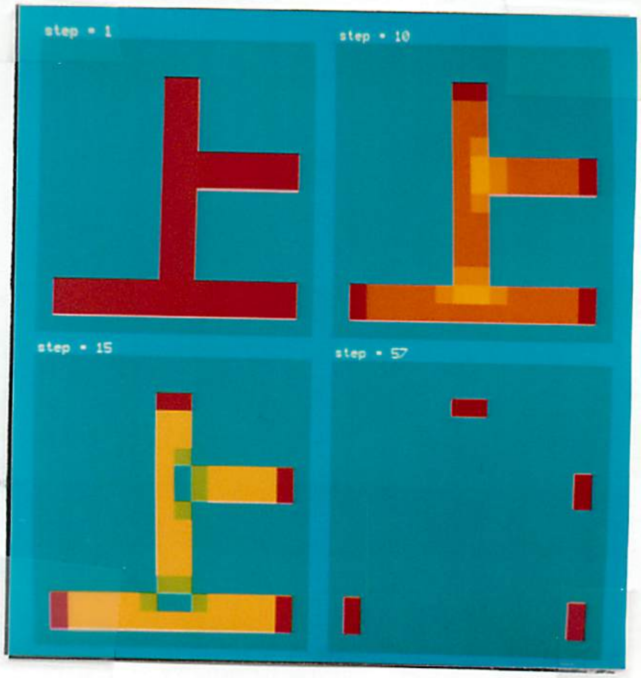


Fig. 3.30

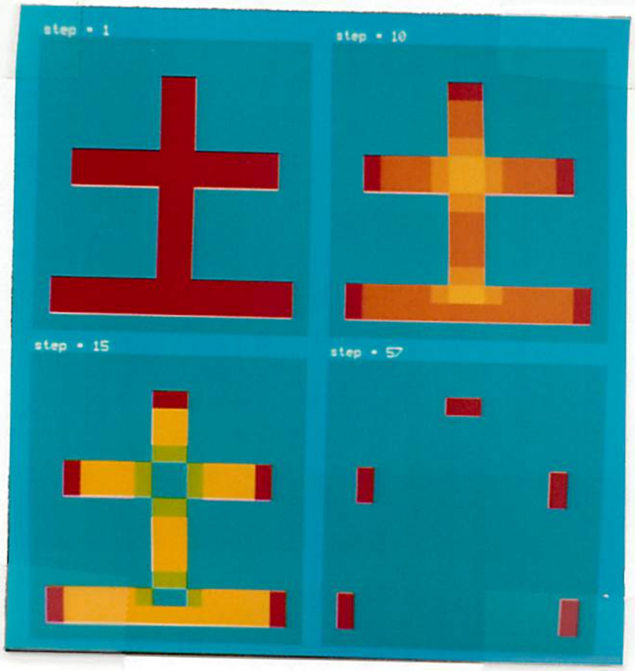


Fig. 3.31

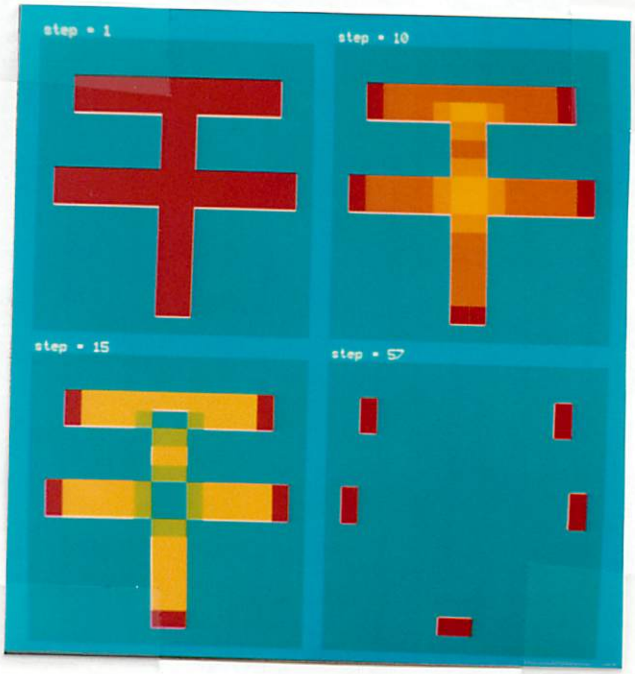


Fig. 3.32

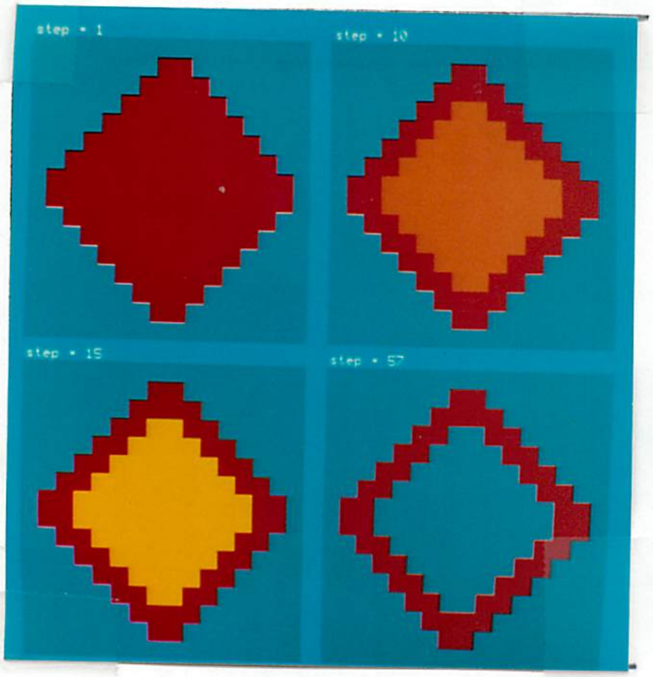


Fig. 3.25

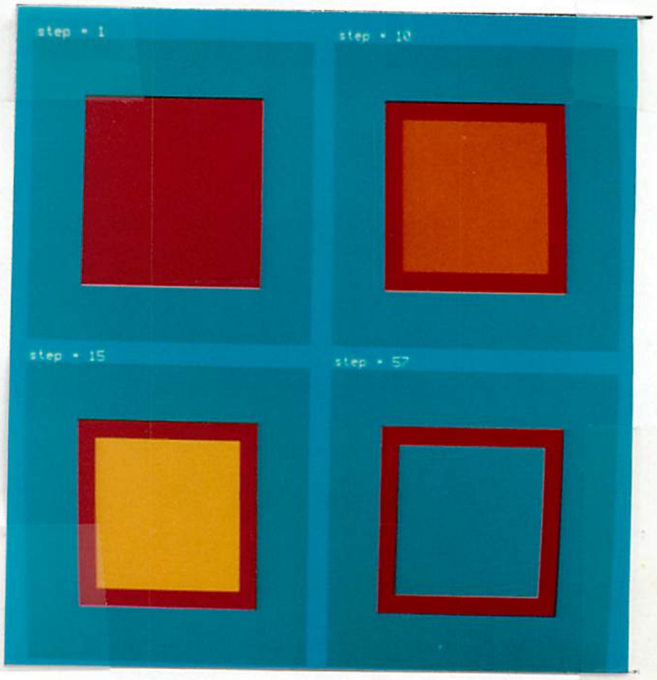


Fig. 3.26

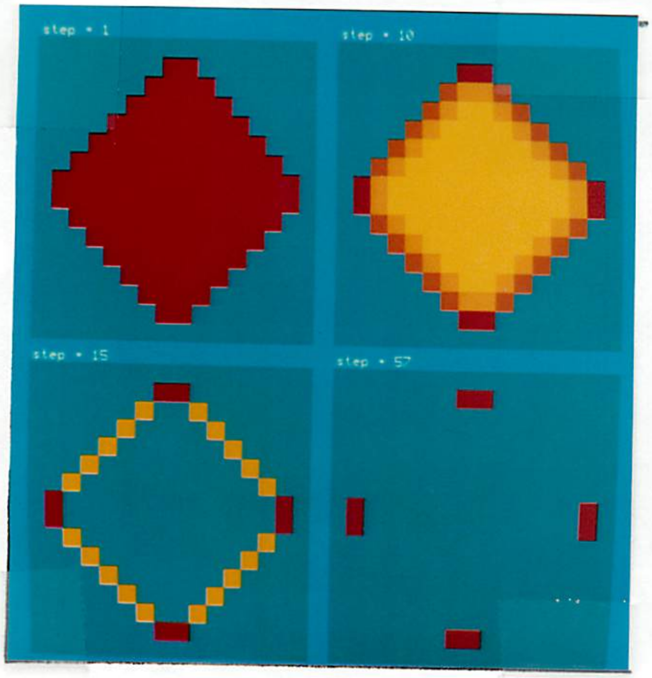


Fig. 3.27

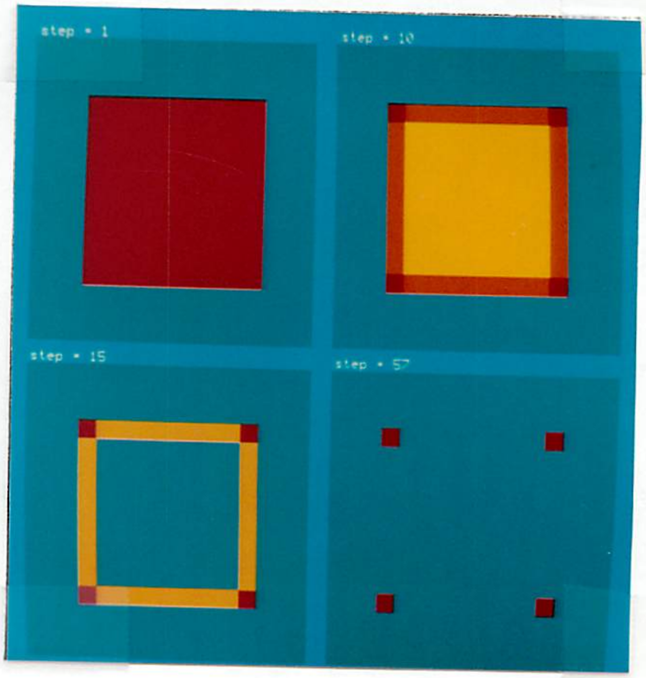


Fig. 3.28

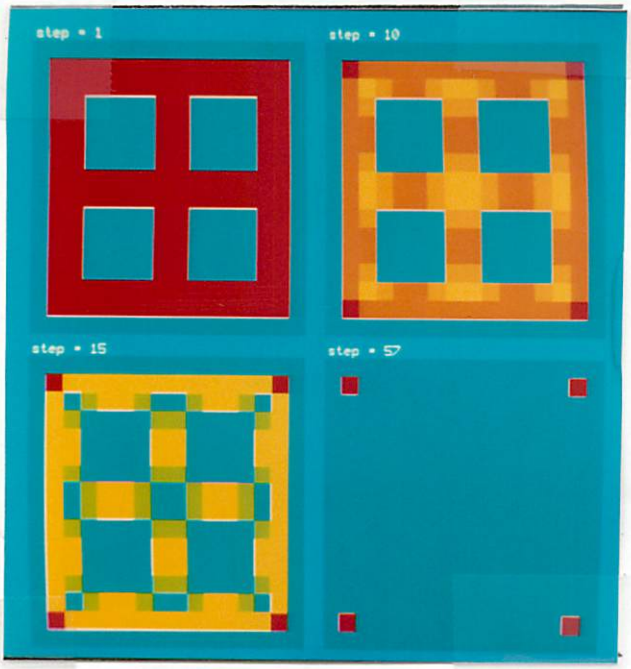


Fig. 3.33

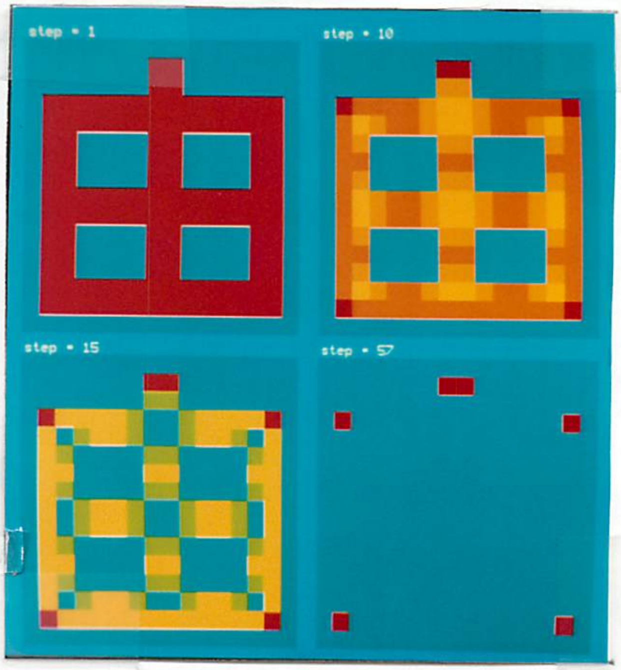


Fig. 3.34

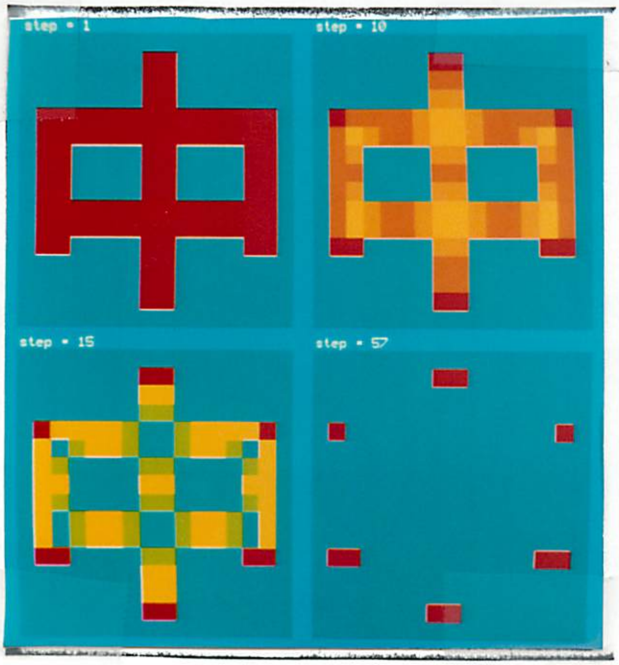


Fig. 3.35

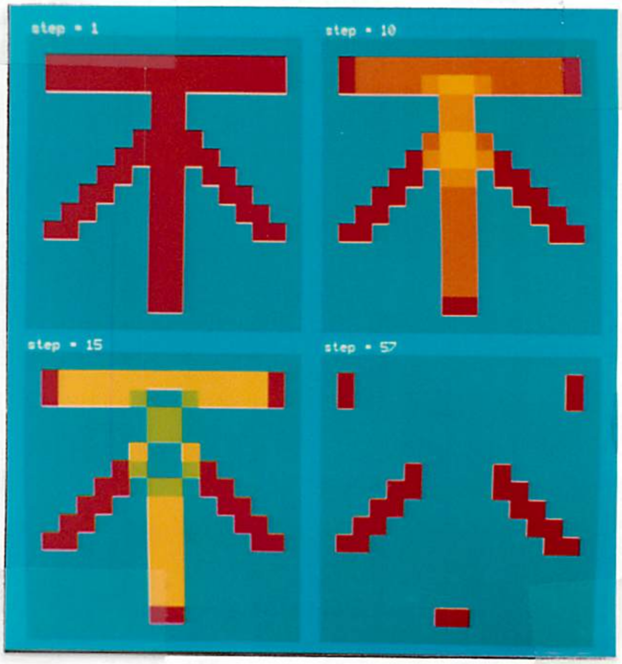


Fig. 3.36