"ON STOCHASTIC MODELLING OF VERY LARGE SCALE INTEGRATED CIRCUITS: An Investigation into the Timing Behaviour of Microelectronic Systems"

Gregory Raymond H. Bishop, B.Sc.(Hons).

Being a thesis submitted

for the

DEGREE of DOCTOR OF PHILOSOPHY

in

The University of Adelaide

(Faculty of Engineering)

South Australia.

January 1993

Contents

List of Figures

"It is plain, then, that it is of ultimate causes that we must obtain knowledge, since it is when we think that we have grasped its first cause that we say that we know a thing.

Now causes are talked of in four different ways: one cause is the being and essence of a thing, what it is for a thing to be what it is (for the reason why a thing is as it is, is ultimately reducible to its definition, and the ultimate reason why a thing is as it is, is a cause and first principle); a second is a thing's matter and substratum; a third is the source of its movement; and the fourth, the counterpart to the third, is the purpose of a thing and its good - for this is the goal of all generation and movement."

ARISTOTLE

Abstract

This thesis is concerned with the statistical design and performance estimation of very large scale integrated circuits.

Variations in fabrication processing parameters cause circuits to operate over a range of speeds bracketing the required design speed. Thus some chips will not meet timing specifications. It is important to economically predict the proportion of manufactured circuits that will meet the design criteria, as this information is ultimately vital for determining whether a given design should be produced at all.

Modern regular design methodologies have given designers the chance of greater freedom of enterprise amongst small and independent applications businesses. To keep the promise of successful and economically viable production of VLSI systems offered to us by these design methods I argue that is necessary to reliably and inexpensively predict the timing statistics of the system under consideration.

Independent designers are inhibited in this regard due to the excessive expense presently incurred in attempting to do this, since about 500 simulations are needed, irrespective of the number of parameters undergoing variation. This contrasts with one simulation to establish the nominal or designed operating speed.

This thesis argues that one heuristic method of solving this problem is by creating new timing simulators which are fast enough because they take advantage of the inherent parallelsim in the operation of the actual circuit.

Part One: Motivation demonstrates the necessity for this design tool by examining the intimate relationship between design yield and the economics associated with the design cycle.

Part Two: Methods completes an study of existing methods which are, or which might be, used to obtain this information, using a fabricated and simulated design of mine as an example. I make a definite rejection of all of these methods as impractical, for a variety of reasons.

Part Three: New Ideas introduces a number of new ideas for stochastic simulation, including: stochastic differential equations, parameterized curve fitting, and an analogue-model structure of mine, with which I had hoped to make a major breakthrough in simulation speed. I am forced to reject these, but the last one leaves a useful legacy, as it leads directly to the idea investigated next, where I concentrate on what I determine to be the only remaining feasible method: a digital method involving a novel algorithm.

My idea is based on the recognition that real transistors operate without global information; their own state and the state of their nearest neighbours suffices for their operation. Thus we might achieve the same sort of fast parallel simulation by exactly the same method, viz., by arranging transistor models in a communicating array, similar to their actual placement in the leaf cell. The implementation of these ideas leads me to a completely new approach from quite a different field, based on notions from economic game theory. This is shown to be reasonable by analysing the abstract nature of the new simulation model. The definition of such a system and the results of simulations are presented.

Part Four: Evaluation assesses the ideas investigated in the thesis in the light of current practice, and considers possible directions for future research.

Declaration

This thesis contains no material which has been accepted for the award of any other degree or diploma in any University, and to the best of the author's knowledge and belief contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

The author consents to this thesis being made available for photocopying and loan, should it be accepted for the award of the degree.

Gregory Raymond H. Bishop

Acknowledgements

I would very much like to thank: Dr D A Pucknell, for his continuing encouragement of research into this field, and for his superb moral support as a supervisor; Dr B R Davis, for his useful discussions with me about statistical analysis; Dr B D Ackland, for his comprehensive review of my thesis; Dr N W Bergmann, for identifying areas that needed extra substantiation; Dr K Eshraghian, for his sustained interest in the results of these investigations; and Messrs M. Liebelt and T. Rohtla for their timely delivery of any amount of computing power, as and when requested.

G.R.H.B.

Author's Publications

N-bit Multiplier

(CSIRO report on AUS MPC 5/82, with P J Whitbread, May 1982)

Analogue Test Structure

(VLSI Program, CSIRO Division of Computing Research, August 1983)

Progress Report: Matching the Monte-Carlo Method with the Continuous Stochastic Dynamical System Model

(Proc. Tri-partite Technical Co-operation Program, Canada May 1983)

A VLSI Architecture for fast integration of certain simultaneous differential equations

(Digest of 'Creating Integrated Systems' Conference, Adelaide May 1983)

Stochastic Modelling for Profitable Design of VLSI Circuits

(Proc. VLSI Pacific-Asia Region Conference, Melbourne May 1984)

An Algorithm for predicting the Statistical Timing Behaviour of Concatenated Structures in V.L.S.I. Circuits

(Proc. 4th Australian Microelectronics Conference, Sydney May 1985)

Statistical Modelling of Microelectronic System Timing Behaviour

(Proc. Microelectronics Conference VLSI 1987, Melbourne, April 1987)

Part I
Motivation

"When the storm rages and the state is threatened by shipwreck, we can do nothing more noble than to lower the anchor of our peaceful studies into the ground of eternity."

JOHANNES KEPLER

Chapter 1

Introduction

To fulfil the promise of successful production of VLSI systems offered to us by regular and hierarchical design methods, it is necessary to obtain the timing statistics of the system under consideration.

One practical method of doing so is by creating new timing simulators which take advantage of the inherent parallelism in the operation of the actual circuit.

1.1 The Thesis

Variations in operating speed of computer systems, arising from fabrication parameter uncertainties, have an intimate relationship to and profound effect on the decision to fabricate the system. The research for acceptable speed in simulation of the timing spread leads to the possible application of ideas and methods quite remote from the field of electronic engineering. It is argued herein that statistical simulation of VLSI circuits, being important for the question of whether a system should be produced, implies the necessity of an array processor architecture for simulating the behaviour of VLSI circuits.

1.2 Why I Consider it Important

The nature of the microelectronic system design field has altered fundamentally and irrevocably in recent years. This is partly due to the rapid spread of design methods, such as those of Mead and Conway [MC80], which have profoundly affected attitudes towards the design of VLSI circuits.

This thesis addresses a subset of techniques for optimisation in Integrated Circuit design, in that it aims at a heuristic, iterative and interactive method that can be used by single workstations.

Digital circuits are so called because their initial stable state and final stable state are all that are of interest - how they got there is usually irrelevant. However, there is a class of digital circuits for which it is of great interest, how they got there, because it tells us how fast they operate. This class interfaces to an unforgiving external world that produces data and demands results at a high and uncontrollable speed.

The search for acceptable speed in simulation of the timing spread leads to the possible application of ideas and methods quite remote from the field of electronic engineering, as we shall see in this thesis.

Available techniques centre around large computers, large companies and large universities [BHSV81]. But how are independent designers to break in, to prosper? This thesis helps to answer to this question, because it investigates the means needed to decide whether to produce a VLSI design.

1.3 Systems Production

Industrial designers of VLSI systems are vitally interested in the question: 'can the system be produced profitably?' Perhaps this ranks equally with the question: 'can the system be designed at all?'

Much recent research effort in Australia has gone into 'how' to design cells, chips and systems; somewhat neglected has been 'whether' to do so. My research into needs in this area has shown that it involves the production of new hardware and software tools for VLSIC design, which are suitable for industrial use on low-cost designer work-stations.

It turns out that statistical considerations dominate the design of tools necessary for answering the profitability question. To create a useful aid to decision-making, we need to combine stochastic models of chip fabrication and of circuit operation with the realistic uncertainties of the marketplace, to form an integrated system.

This thesis investigates many proposed methods and discusses some ways of implementing possible tools in a practical way on small systems.

1.3.1 Missed Opportunities

There are systems that are not produced because it seems they are not profitable, but that actually should be produced, because they are profitable; and there are systems produced that seem profitable, but which are actually not. Both are bad, and both can be avoided by a "statistical" design.

1.3.2 Expectations

For the practical designer, as distinct from the researcher, who really will want a large number of chips from a design- fabrication run at \$40000-\$50000 per run, a very important consideration must be: what is to be the proportion of good chips? When designing to customer specifications this is vital. However it is clear that variations in process parameters for each layer, and variations in actual operating conditions, will mean a spread of timing behaviour in the chips. How can these myriad effects be assessed, and thus the need for a re-design determined?

The most important tool for this is SPICE or SPLICE or one of the much faster recent tools. However, even with the faster programmes there is an inherent problem - they either take too long for this purpose, or they are too inaccurate, as will be seen.

Maly [Mal90] describes techniques that can be employed not only to design a chip so that it meets a set of functional specifications, but so that it will have adequate yield when manufactured. He urges the premise that CAD in the future must be concerned with the entire product life cycle, from specification through qualification. Along these same lines is the growing feeling that to be competitive it is important to take an integrated view of CAD and CAM. This view is motivated by the fact that actual success of a chip design depends on the ability to manufacture the chip with sufficiently high yield to make it profitable.

Strojwas and Sangiovanni-Vincentelli [SSV86] in their review of statistical techniques, feel strongly that the relevance of the field of stochastic design is, if anything increasing, and that the bottleneck is still circuit analysis. Yield maximisation approaches have to be extended to cover all significant causes of failure, and have to be supported by physically based models. Research efforts should be focused on economic aspects of VLSI design, manufacturing and testing. New approaches should integrate all these considerations.

1.4 Stochastic Modelling

Maly, Strojwas and Director [MSD86] point out that, due to inherent fluctuations in any integrated circuit (IC) manufacturing process, the yield, nominally viewed as the ratio of the number of chips that perform correctly to the number of chips manufactured, is always less than 100 percent. As the complexity of VLSI devices decrease, the sensitivity of performance to process fluctuations increases, thus reducing the manufacturing yield. Since profitability of a manufacturing process is directly related to yield, the search for computer-aided methods for maximising yield through improved design methods and control of the manufacturing process has intensified dramatically. Statistical approaches to yield modelling and optimisation have been under development for a number of years. For the most part, these methods can be separated into two categories: parametric yield estimation and optimisation techniques and catastrophic yield estimation and optimisation techniques. In general, parametric yield optimisation has been formulated as a tolerance assignment or design centring problem. However, due to simplified assumptions about circuit element characteristics, these approaches have been proven successful only for discrete circuits with a relatively small number of designable parameters. All of the physical phenomena that affect manufacturing yield are taken into account.

Spanos and Director [SD86] stress that the IC manufacturing process is subject to inherent statistical fluctuations of material quality and equipment performance. Characterisation of these fluctuations is important if performance of the manufacturing facility to be simulated and optimised. The IC process consists of at least 3 distinguishable hierarchical entities, namely the chip, the wafer, and the lot of wafers, and variations should also be modelled hierarchically.

Mei and Dutton [MD83] also feel certain that technology modelling will become increasingly important in future VLSI fabrication, considering that typical VLSI circuits consist of more than 100,000 transistors on a single chip less than $1cm^2$ in area with the minimum device feature size on the order of 1μm or less. Models for simulating both lithographic and etching steps have been subjects of intensive research, since many physical parameters such as grain size and active concentration of dopants change during processing, and consequently modify other processing constants such as diffusion constants and oxidation rates.

Benkoski and Strojwas [BS87] note that simulation is the best means to verify the behaviour of VLSI circuits from both the logic viewpoint and from the timing viewpoint, but that, unfortunately, while strict hierarchical design methodologies have been developed in order to manage the complexity of the design process, the simulation task has not benefited from the same attention. In addition, in today's technology, the assumption that process parameters are constant between dies, or even within a die, is no longer valid. As a result, statistical verification of the timing is now sought. Since statistical analyses require repeated runs, the cost of each simulation is even more crucial. Moreover, in order to perform a meaningful analysis of the variations of the process parameters, the simulation accuracy must be further improved.

Yang *et al* [YHC$^+$86] echo this when they identify the major problem in statistical circuit design for MOS VLSI as the prohibitively expensive computational requirements.

Scaling of feature size progressed more rapidly than scaling of process tolerance. So at micrometer and sub-micrometre geometries used now in VLSI, statistical variations of device characteristics can be very significant. These variations in device characteristics result in corresponding variations in circuit performance and must be considered in VLSI design.

Lightner [Lig87] appreciates that simulators are used for a variety of reasons; because systems implemented in silicon cannot be effectively breadboarded, to produce a verified specification of the system performance, to verify performance of portions of the design against specification, to test the design for possible failures.

However, Saleh *et al* [SGC+89] remind us that circuit simulation is a very time-consuming and numerically intensive application, especially when the problem size is large as in the case of VLSI circuits. The time-domain transient analysis is the most computationally expensive in terms of CPU times. Programs such as SPICE2 were originally designed to simulate circuits containing up to 100 transistors, but have been routinely used at some companies to simulate circuits containing over 10,000 transistors, at great expense.

The problem is that these programs take a long time to run. That might be viewed as simply an essential thing to put up with, but there is a more important and more subtle consideration to be noticed: that very fact of a long time deters designers from investigating the effect of all errors and tolerances in parameters.

1.5 A New Approach?

It seems that a fruitful new approach might be possible, based on a new approach to the simulation process. In fact, after examining a number of software and hardware methods for speeding circuit simulation, a pseudobreadboard idea occurs and is examined, but needs to be discarded. However, it leads to a related digital method that is full of promise, especially in view of current trends and advances in wafer-scale integration. Examination of this derivative idea comprises the bulk of this thesis, and is not without surprises.

1.6 Unaddressed Problems

1.6.1 Parameter Variations

No small system software tool addresses the effect on a circuit design of parameter variations, even though they are an inescapable aspect of the design and fabrication cycle. There are no typical parameter variations: the range of values depends mainly on the fabrication houses, once the process has been decided on.

1.6.2 Longest Path-Criterion

The so-called 'worst-case' or longest-path time is often extracted from leaf cells by circuit simulation and used in the hierarchical simulators. But is it the relevant time to extract? Along the longest path, there are wide process variations which will make even the longest path even longer (or shorter). Which time should be taken? To answer this requires statistical simulation.

1.6.3 Example of Timing Variation

Modern nMOS circuits use clocking frequencies of 10 to 15 MHz, which means that the combination parts between clock signals are working in times of about 60 nanoseconds up to about 100 nano- seconds. To see what we have to contend with, consider an actual example of a leaf cell used in a multiplier structure, described in Chapter 3.

Using nominal values of capacitance, resistance and threshold voltage, the nominal longest path is 51nSec. Using the extreme variations of these three parameters yields times of 33nSec and 75nSec as the maximum and minimum times. In order to run at 15MHz (66nSec period) redesign of the cell appears mandatory; in reality, however, there is no indication of whether this is true or not; perhaps only 5% of all chips would take longer than 60nSec: a stochastic simulation would be needed to show this - there is simply no other way of doing it. This is important because re-design is time-consuming (because all stages of the CAD system have to be revisited) and will probably result in greater power consumption (because the chip has to work faster). Re-design in the case of this example may now be a waste of time because 95% may be to specification; the *nominal* and the *worst case* approach to design can often be simply misleading.

On top of this there are cases where cells of this type are cascaded into larger structures - what are the mean and spread now? Does the usual high-level simulator provide the answer easily? Is it important to know the answer? Chapter 6 develops a possible way around this.

1.7 How to Simulate - A New Way

What is to be simulated? To build up the timing statistics of the system, it is necessary and sufficient to obtain the timing statistics of leaf cells on the critical timing path. Figure 1.1 on page 10 summarizes what is expected in this regard.

In this thesis we shall see what turns out to be a practical and, under some circumstances, a fast approach: we attempt to 'breadboard' the VLSI circuit, and much of Part III is concerned with showing that this is feasible.

Part IV argues that the algorithms developed in Part III have a wider significance.

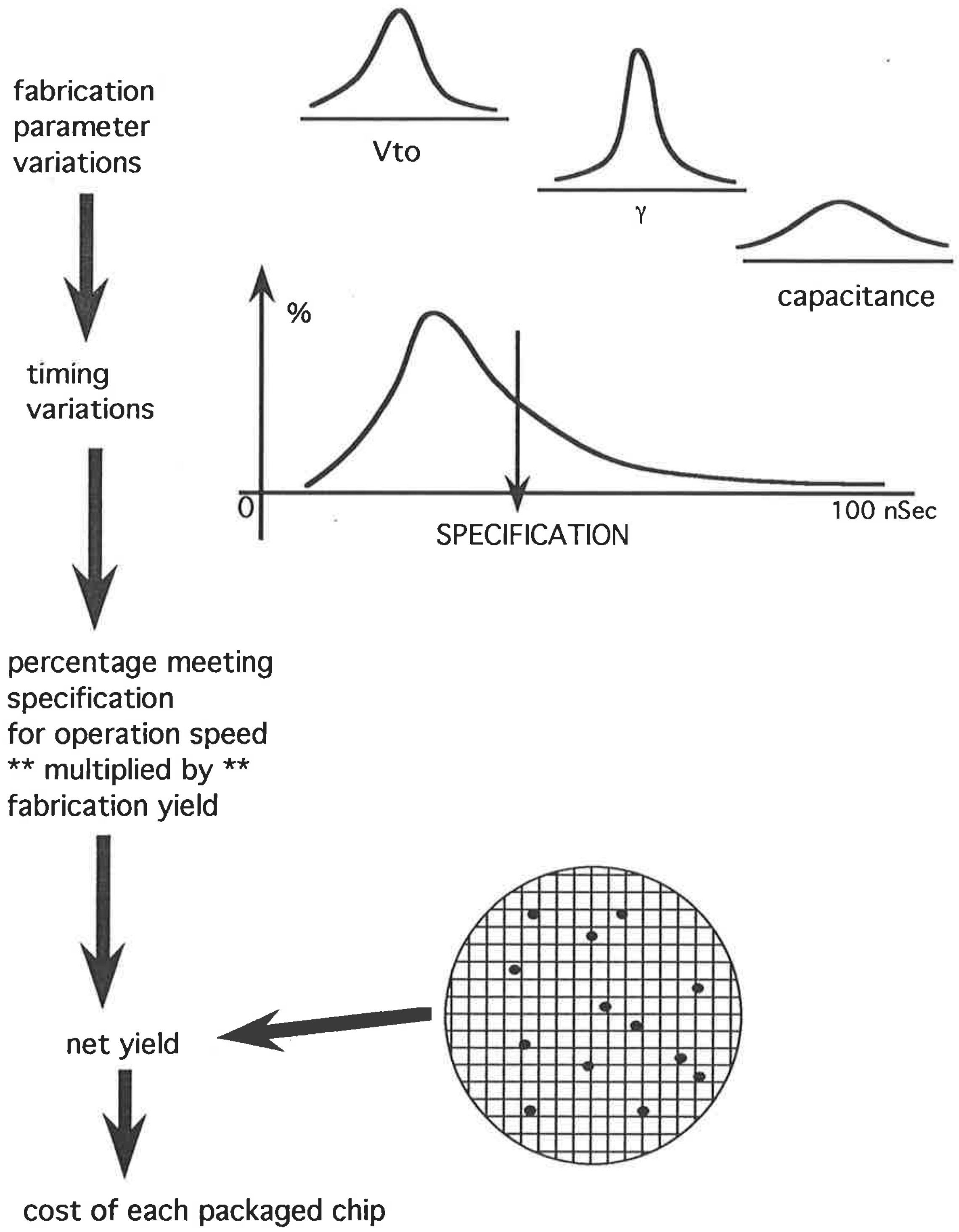

fabrication parameter variations
Vto
γ
capacitance
%
0
100 nSec
SPECIFICATION
timing variations
percentage meeting specification for operation speed ** multiplied by ** fabrication yield
net yield
cost of each packaged chip

Figure 1.1: Overview of Net Yield - How it is Obtained

Chapter 2

Economic Considerations

In this chapter I discuss the important influence of economic considerations on the design cycle.

2.1 Statistical Considerations

The most difficult decision faced by the independent designer group is whether to proceed to fabrication and production of their design. Myriad compromises are involved and the highly non-linear and discontinuous nature of the problem makes the only practical method one which maximises the probability of success by running hundreds of scenarios and compiling their statistics. This is true in the area of design, fabrication and marketing.

The greater the range of proposed systems that can be evaluated, the better for everybody. The cost of evaluation is extremely high, particularly of redesign, which fact favours entrenched companies. If the cost of deciding whether a design should go ahead can be made negligible then an important barrier to the entry of newcomers is removed, and more rapid progress through extra production and experimentation can take place.

2.2 Profitable Design of VLSI Circuits

Mozumder [MS90] emphasizes that with increasing complexity, VLSI circuits and process technologies are being pushed to their limits, and fluctuations in IC manufacturing are becoming the predominant factor of profit loss in fabrication lines.

Riley and Sangiovanni-Vincentelli [RSV86] also note that what has limited the rewards of shrinking device dimensions is not any set of deterministic relationships, but the variations in realised device dimensions from circuit to circuit which results in variations in performance, which in turn significantly degrades the economic value of the totality of circuits produced.

The aim of any VLSI manufacturer is to maximise the total profit while meeting all design constraints on the product. The profit is affected by a number of factors. The inherent process fluctuations cause significant yield loses, and this results in a drop in the profit.The total profit is affected by the different fabrication and assembly costs and the revenue associated with fabricated IC's. These costs and profits are in turn dependent on several aspects of the particular fabrication process that the IC's undergo. Some of the processing steps are performed on entire lots, and hence the total cost associated with such steps will not depend significantly on the number of wafers. Other processing steps are performed on individual wafers and the costs associated with such steps depend on the number of wafers. After the wafer processing stage, the wafers are divided into individual chips and these are probe-tested. The chips that pass these tests are assembled and packaged into IC's. These IC's then undergo functional tests.

Thus it is evident from the profit maximisation framework that the design, control, and diagnosis phases of a VLSIC are strongly coupled.

2.2.1 Market Yield

There are many decisions to be made during the design phase, as indicated by Sze [Sze83]. The selling price of the system is a function of the number and nature of the desirable features contained within the system (on the chips). The die size is determined by the number and complexity of these features. The size of the fabricated die is determined by the technology. The cost of each die is set by the technology, number of wafer starts and wafer yields. The yield is strongly determined by the size of the die. The ultimate cost of the system is determined by the cost of each die and how many may be sold. The ultimate returns in the investment are determined by the excess of selling price over cost and the number sold. The number sold is partly determined by the selling price and the number of desirable features . . .

Thus a full circle is turned: whether the system should be produced at all only makes sense if the number to be made is estimated. This gives the cost of each possible course of action. But the number to be made only makes sense if it is asked: what is to be done with them? How many may be sold? At what prices? Thus the probability of events remote from the design phase itself appear to be vital for practical exploitation of VLSIC products. It is clear that a more comprehensive, integrated view of all phases is needed.

2.2.2 Design Yield

One way to manage the complexity of the VLSI structures is to design hierarchically.

Circuits may be considered as being composed of leaf cells [MC80], and if the leaf cells are properly characterised then much may be deduced concerning the statistical behaviour of the circuit and system timing. By knowing the characteristics of the leaf cells, and the interconnections between the leaf cells, one can perform timing verification and obtain the characteristics of the composition cell.

Timing estimates are usually obtained for the nominal operating conditions and for the worst case. Neither of these times sufficiently characterise the circuit behaviour for fabrication purposes. These two figures give the designer almost no confidence at all when predicting the actual performance of the chip.

The problem with running only the nominal case is that no idea of the possible spread of results is obtained; if the spread is large this is a very dangerous omission. Even if it is small, greater confidence in the design is assured if the spread is known.

The problem with worst-case methodology is that it is performed with an extreme set of parameters which renders the predictions unnecessarily pessimistic. Truly reliable predictions may only be made if all the variations are taken into account from the start of the simulation. The greatest uncertainty about actual performance arises if these parameter variations are ignored, and only '*exact*' values are used; this is worse than useless when vast sums need to be committed for chip fabrication.

If stochastic simulation shows that (say) 50% of all returned chips should meet specifications then the cost of each chip is known (from the manufacturing yield and the design yield). If this cost is unacceptable then the leaf cells of the chip may be redesigned so that a greater percentage of chips will meet the specifications; this makes a larger chip and leads to a lower yield (using the same process). Only a model which takes account of all costs and profits in a statistical fashion can show whether the redesign was useful or necessary.

2.2.3 Fabrication Yield

The fabrication yield from a run is needed to obtain the final yield for calculating chip cost. There a many formulae available from various sources which give an estimate of the mean yield based on the process and the chip area. An extension of the above example might point this up: consider a design yield from stochastic simulation of 50% and a fabrication yield of 30%; this gives a net yield of 15%. Suppose the cells are redesigned to give a design yield up to 90%, leading to a larger chip with perhaps a fabrication yield down to 15%: the net yield now becomes 13.5% which is a little worse overall! A redesign of the architecture itself might now be indicated.

It is clear that the need to obtain these figures implies a mathematical model of greater scope than normally associated with chip design alone.

Peltzer [Pel83] raises the possibility that wafer-scale integration (WSI) can improve system reliability, by the use of redundant circuits, and thus reduce the yield loss caused by small random defects; however, WSI demands a sizeable investment in computer-aided design.

McMinn [McM82] remarks that, as feature sizes have reduced, the size of dies have doubled with no loss in yield; this means that designers can dramatically increase the amount of circuitry. The tradeoff between die size and projected yield is an important consideration when partitioning the system for a custom chip. eg for a typical (then!) 5 micron silicon gate nMOS process with 4-inch wafers and 5 critical mask levels , slightly more than doubling the area causes a near 6-fold decrease in yield.

Tsaur and Chen [TC86b] demonstrate the spread of processing parameters in very small feature-size CMOS devices, with effective channel lengths of 0.7 to 4μm. The threshold voltages are found to have a 99% spread of 240mV. The propagation delays are about 95psec at 5V, which are acceptable ranges for 5V devices, but would have to be reduced for lower voltage and submicrometer devices.

Styblinski and Opalski [SO86] have developed a set of tools. Their premise is that, as ICs become increasingly complex, and geometries smaller and smaller, it is becoming more and more difficult to achieve acceptable manufacturing yields, even if the normal design fulfils all design constraints. The manufacturing yield is composed of two parts: the technological yield and the design (or parametric) yield. The former is a result of catastrophic failures. The latter is a result of the sensitivity of circuit performance to IC device parameter variations, caused by unavoidable variation of the manufacturing conditions from device to device and from chip to chip.

2.2.4 Manufacturability

In a very important and wide-ranging paper, Maly [Mal89] explains the concept of design for manufacturability, and lists tasks and CAD tools dealing with manufacturing aspects of the design of modern VLSI circuits.

Traditionally, computer-aided design (CAD) tools are applied to create a nominal design of a VLSI circuit - the design which meets desired nominal functional specs. In reality, however, the nominal design, along with the manufacturing process, must be very often modified to maximise manufacturing yield. Such maximisation must be performed during the design to achieve an acceptable level of initial manufacturing yield. It also must be performed during fabrication in order to achieve the maximum rate of yield improvement in the entire product development cycle.

A typical approach to the design of VLSI circuits, and IC design is produced by using given "nominal" characteristics of the VLSI circuit elements. In the general case, however, they are random and may be so large that some of the fabricated ICs may have unacceptable performance.

Manufacturing yield[1] is directly related to the manufacturing profit - the most important figure of merit of any manufacturing activity, because it is a measure of the incentive to reinvest in the product line.

In general, the VLSI product development cycle must be seen as a process of a number of decisions performed iteratively by using uncertain information: the first is a necessity to predict performance of the IC using incomplete and uncertain data; the second is the existence of random environmental factors which disturb prototyping and manufacturing steps. In the modern/fabrication facilities, the vast majority of defects are small (in the range below 2 μm).

For a new product planning-phase one must decide: which technology should be chosen and what should be the die size, expressed in terms of the number of transistors and the area of the die? In the case of a system

[1]the ratio of fabricated IC chips with acceptable performance to the total number of produced chips

composed of a number of different IC's , one may also ask whether the system components should be fabricated on a number of small dies or on one, large-area, common substrate?

Usually the answer to such questions is not trivial because it involves a number of not very well-defined tradeoffs, as well as various competing objectives. In general, however, one can assume that the optimal technology and the size of an IC are such that they result in a maximal manufacturing profit. But the profit is a function of the yield which is, in turn, a function of applied design rules and the die area. Consequently, in order to determine the economic viability of a new IC one must know relationships between design rules, die area, and yield, and the relationship between yield and manufacturing profit, for all available technologies.

The key to the strategic design decisions is yield prediction, which can take into account: the actual characteristics of the process disturbances causing yield loss; the relationship between yield and topology of an IC layout, with a special emphasis on geometrical design rules; and yield improvements that can be achieved through the application of various redundancy strategies. None of these three aspects of yield prediction can be handled without the aid of the computer.

One of his major conclusions is that a yield model must express yield losses in terms of the die size and IC layout characteristics rather than in terms of the IC area alone. Such a need becomes more and more pronounced with the decrease of the minimum feature size of modern technologies.

In particular he notes that there is more and more evidence that large-area integrated circuits, including WSI, will soon become economically feasible, and concludes that design for manufacturability through the application of redundancy techniques that are becoming established in the WSI field seems to be very promising and in the future should be used much more often. But this will require better yield-prediction tools, that can be used early in design cycle, and which are accurate enough to predict specific modes of circuit malfunctions well. The problem is that smaller IC elements are more sensitive to inherent fluctuations of the device dimensions, doping levels and so on.

Thus Maly considers that key components of a set of yield tools should therefore be: process simulators, device simulators and yield and performance predictors. A whole spectrum of methodologies have been developed to solve the yield maximisation problem, but a majority of them have a circuit-optimisation bias, and therefore have been developed on a strong theoretical basis, with less stress on the process realities and manufacturing-related constraints. A special group of theoretical approaches form the methods using Monte Carlo techniques to estimate yield and to perform yield optimisation as well.

The most important point he makes from the perspective of this thesis is that, although circuit simulation tools are the oldest amongst CAD tools, they still form a major bottleneck to design for manufacturability. This is because parametric yield evaluation requires full accuracy of simulation, and therefore simulators matching the complexity of SPICE must be used.

He concludes by noting that a long list of theoretical and practical problems must be solved before manufacturability-oriented CAD tools can be used in anyone package, mainly because of a traditional split among circuit/systems design, process development, and manufacturing mentalities; but also because many theoretically elegant but inherently inefficient algorithms and procedures cannot be directly applied to VLSI circuit/process design because of the size of the modern VLSI circuit.

2.3 New Design Tools

Variations in process parameters and actual operating conditions will mean a spread of timing behaviour in the resulting chips. New tools are required to assess this spread and hence help to determine of a redesign is necessary.

The most important tool for this a a leaf-cell circuit simulator. All methods eventually aim at producing the only truly significant statistic: the joint accumulated distribution function. This shows the proportion of leaf cells which have completed operation at any time, given the process variations. Thus it also shows the yield of the chip design in terms of the timing operation specifications. This may be combined with the fabrication yield to give the net yield of the design itself.

However, the problem is subtle. In IC fabrication processes, yield maximisation, as attempted by the quality control system, does not imply profit maximisation. A lot that is rejected at an intermediate stage results in zero yield but saves the cost of future processing steps and frees up the fabrication equipment. The freed-up resources could then be used to produce a lot with a much higher yield. Therefore the objective of the control system will be profit maximisation as opposed to yield or though-put maximisation, whereas the objective in design for manufacturability is to develop a design that minimises the sensitivities of the output performances of the fabricated IC's to the random disturbances affected the manufacturing process. These are slightly different.

Thus Mozumder [MS90] hopes to achieve profit maximisation using a unified framework where both quantity and quality controls are used as part of statistical process control on a fabrication line.

Riley and Sangiovanni-Vincentelli [RSV86] point out that the reason that expected economic gain is not usually the explicit criterion for engineering design selection is that an explicit economic model is generally not readily available. Nevertheless, inclusion of explicit economic models in engineering design is the ideal, and a truly advanced design methodology should incorporate such models. Fortunately, incorporating revenue models, which are heavily based on statistical methods, into the design methodology does not introduce the need for radically different knowledge bases or software capabilities, once the problem is already statistical in nature.

Spoto, Coston and Hernandez [SCH86] have created a Statistical Analysis Menu which integrates process, device, and circuit characterisation functions. Their opinion is that the increasing competitiveness of the IC industry will eventually dictate the use of such an integrated system on all product and cell developments in order to squeeze out as much performance and area as possible, and in particular to avoid overdesign of products.

Gallivan *et al* [GJMW91] have noted that in recent years, all U.S. supercomputer manufacturers, and dozens of mini-supercomputer vendors have entered the market with some form of general-purpose parallel-processing system. This might be a good thing from the point of view of faster simulation, but judgement should be reserved and this issue is addressed at some length in Part IV.

2.4 Some Observations

CAD for VLSI circuit manufacturability must be able to cover a large variety of design problems on all levels of design abstraction generated by all phases of the device development cycle.

The first important observation indicated by the discussion above is that manufacturing CAD, which is IC performance-oriented, is perhaps more than any other kind of CAD affected by limitations of available computers. Such design tasks, involving simulation or optimisation steps, performed on the low levels of design abstraction (transistor and below), are very computation-intensive and not all of them can be solved with the existing tools and available hardware.

Computing power capable of producing the necessary statistics is, in large companies, already devoted to projects they judge important. Such power will cost a lot to divert to speculative ventures.

If the promise in the computer circuit revolution initiated by the new design methodologies is to be realised, then even slightly-capitalised groups need to be able to enter the field. The greater freedom to cheaply determine whether a design should proceed to market means that both a greater range of possible systems can be tried and that once it is determined that a system may be economically produced, then existing large capital groups can become involved. However, up to that stage they tend to inhibit new developments, which the main reason that the ability to try new methods should be available to the poorest (in terms of computing power) as well.

This need cannot be ignored in the era when VLSI is approaching ULSI, and when almost everybody understands that the true engineering and economic achievement is not merely the design which can be successfully simulated and tested on the first silicon, but the design which can be easily manufactured in large quantities - an IC designed for a high level of manufacturability.

2.5 Significant Conclusions

This thesis presents results of importance to independent designers of VLSI circuits and systems, especially those who are not associated with a large company or do not have access to large computing facilities.

Profitable production of VLSI chips and systems depends on net yields from design and fabrication, amongst other factors. Increased yields may be obtained by using a smaller feature size or a different design algorithm. However, the effect on profits of either course of action cannot be assessed without adequate mathematical models, and these have to be included in the design cycle itself.

It turns out that statistical considerations dominate the design of tools necessary for answering the profitability question. To create a useful aid to decision-making, it is necessary to combine stochastic models of chip fabrication and of circuit operation with the realistic uncertainties of the marketplace, to form an integrated system.

On single-user designer workstations, this set of very complex and interrelated constraints might conveniently be investigated by simple, heuristic and interactive methods, using accurate stochastic models. Such models are investigated in the Part III.

Part II
Methods

"Philosophers are as free as others to use any method in searching for truth. There is no method peculiar to philosophy."

KARL POPPER

Chapter 3

Circuit Timing Characterisation

Here I explain what is meant by an adequate characterisation of the timing behaviour of a circuit . This is illustrated by completing a statistical simulation of a leaf-cell from an already-fabricated circuit .

3.1 Types of Circuits

There are many aspects to circuit characterisation, such a logical behaviour, timing and waveform behaviour, density of packing, the manufacturing technology (nMos, cMOS, hMOS, vMOS, &c.), the feature size (5,4,3,2,1 microns and below), and power dissipation. However, as far as predicting whether or not the circuit will work the important characterisations are those of logic and timing. Logic has been well explored elsewhere, in the literature and in practice, and attention here will be on timing.

The relative importance of logic or timing simulation depends on the circuit under investigation.

Clocked Systems In clocked systems, combinational logic occurs between clock lines and is almost exclusively the subject of investigation, and both logical behaviour and timing are crucial to the success of the design.

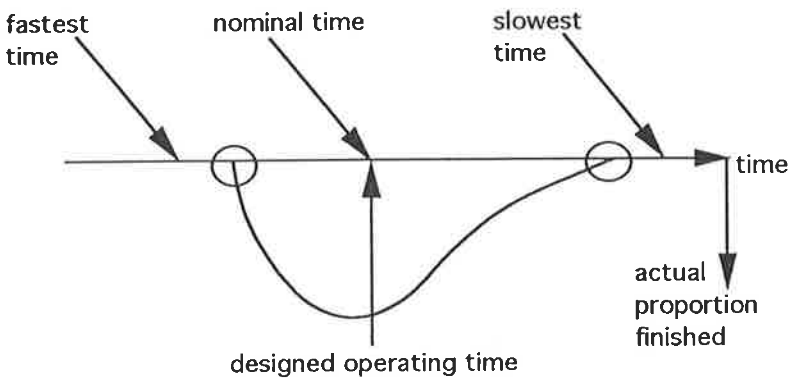
Data Path Systems These systems are unclocked and it is the logic behaviour that is paramount in the first instance. However, the status nodes which wait for a 'true' or 'ready' signal from all inputs (from processing elements) are effectively 'clocks'; all transient behaviour waits for them to 'fire' and so even data-path systems are a subset of clocked systems, in this sense. Thus once again it is the behaviour of processing elements between nodes which becomes important. This covers "handshake"-operated sequential (asynchronous) logic, too.

Cascaded Combinational Logic Systems These, such as array multipliers, are also unclocked, and once again logical behaviour is required to be verified and timing predicted, so that stochastic studies are important.

3.2 Timing Simulations

Timing estimates used to be made by hand using the TAU model. However, with circuits now held in a database it is much more convenient and accurate to extract the circuit and submit it to computer simulation using programmes such as SPICE [NPSV89], SPLICE or QRS [Poi83]. In doing so it is typical to consider the 'worst-case' for process parameters. The result of the simulations is a time of operation for the nominal operating conditions, and a worst case time.

However, as has already been seen in the discussions of economic justification, because many of the important parameters that determine the timing behaviour are not tightly defined, this will often not be the best or even a useful characterisation. The slowest time of operation is obtained using extreme parameter values that, taken together, produce an extremely unlikely combination in practice. In this sketch



the fastest operating time, on the left, falls somewhat below the fabricated fastest time, by an unknown amount that can only be predicted using a stochastic simulation. So instead of the complete distribution curve, shown below the time axis, the only values available are the fastest, the nominal and the slowest times.

Since the actual nature of the distribution function is unknown, and in particular the *circled* fabricated times are unknown, these three figures by themselves give the designer almost no confidence at all in predicting the actual performance of the chip at the designed speed.

3.3 Leaf Cells

Considering that all the main types of circuits may be composed of leaf cells, then if the leaf cells are adequately characterised statistically much may usefully be said about the statistics of the circuit containing them.

A leaf cell typically has a small number of inputs and a small number of outputs. A change in the voltages at the inputs (consequent on clock or ready signals) cause a change in the outputs. This change takes time – a different time for different values of process parameters such as feature size, transistor size, threshold voltage, capacitance and resistances of layers of metal and silicon.

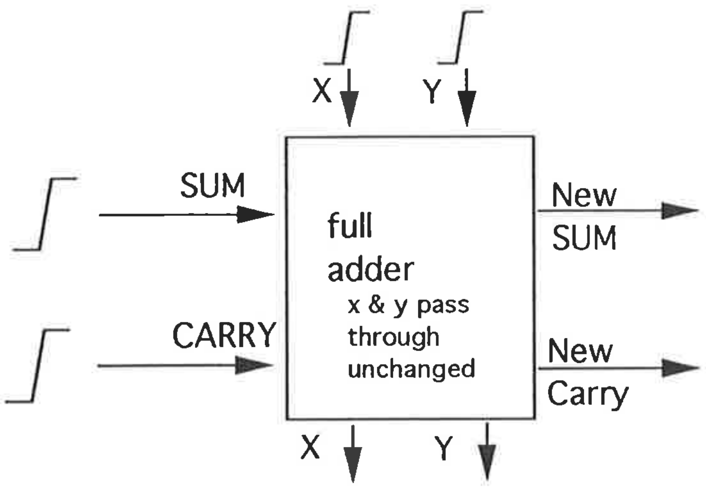
The prediction of this time is needed for a redesign decision, but it is not a *definite* time of operation; rather, there is a spread of possible times due to the spread of possible parameter values during fabrication. The greatest uncertainty about actual performance arises if these parameter variations are ignored, and only 'exact' values are used. This is useless when large sums need to be committed for chip fabrication. Thus the importance of a full stochastic simulation – it is only if all these variations are taken into account from the start that truly reliable predictions may be made.

In addition to the single-cell case, there are cases where suitable cells are concatenated to form larger structures, such as ripple adders and their like. What is the mean operation time and what is the spread of times in this case? Downs [DCR82] does not think that the usual high-level simulators can easily provide the answer. This problem is addressed in chapter 5.

3.4 Monte-Carlo Simulation Example

Typical nMOS circuits, available when the bulk of the present simulations were carried out, use clocking frequencies above 10 MHz. This means that the combinatorial parts of the circuits between clock signals are working in about 100 nanoseconds.

An actual example of a leaf cell used in a multiplier structure ,designed by the author, is fully characterised by Monte Carlo methods to demonstrate the desirable and attainable results of stochastic simulation. The simulations are all carried out on a IBM 370/3033 *CPU* using the SPICE 3B.1 program [NPSV89]. The leaf-cell is shown in Appendix A. It has been fabricated, tested, and it works ['N-bit Multiplier', Author's Publications]. Shown below, the *sum, carry, x & y* inputs pulse from 0 to 5 volts and the cell produces new *sum* and *carry* outputs. Operation finishes when both the *sum* and *carry* reach a high of 2.4 Volts.



3.4.1 Fabrication Parameter Variations

It is usual to calculate with variations around the nominal values out to the 3σ or three standard-deviation point. For a gaussian distribution this means that practically 99% of all possible values lie within two extremes. Variations in some of these quantities are correlated during fabrication, [Coh83] [BS87] [CYC84] [DCR82], in which case the correlations are incorporated into the simulation by the sampling technique described in Appendix B. The table overleaf

PROCESS PARAMETER	NOMINAL VALUE	MIN. 3σ	MAX. 3σ	CORRELATION COEFFICIENT
enhancement V_{to}	0.85 V	0.6 V	1.1 V	0.3
depletion V_{to}	-2.75 V	-3.4 V	-2.1 V	
enhancement γ:	0.62	0.57	0.67	0.1
depletion γ:	0.65	0.60	0.70	
sheet resistivity: *(ohms per square)*				
diffusion	10	0.9	1.1	0.2
polysilicon	20	0.9	1.1	
metal	0.030			
line width: metal	1	0.9	1.1	0.2
line depth: metal	1	0.9	1.1	
line width: polysilicon	1	0.9	1.1	0.2
line depth: polysilicon	1	0.9	1.1	
capacitance: *(pf per sq micron)*				
diffusion to substrate	0.000010	0.9	1.1	0.2
metal to substrate	0.000003	0.9	1.1	0.2
polysilicon to substrate	0.000040	0.9	1.1	

summarises the nominal values and the spread of the process parameter values used in the simulations. These values are for simulation purposes only. Although they are based on some parameters that were assumed for a Multi-Project Chip fabrication, they do not correspond to any actual process. Correlations are assumed between bracketed neighbouring pairs of parameters in the table. The simulations are carried out at a fixed temperature of 27.0 degrees Centigrade and a fixed operating voltage of 5 Volts.

3.4.2 The Longest-Time Path

First the input 'decks' for 256 SPICE runs by editting the voltage input 'card' on the nominal deck. This allows the four inputs to change from all combinations of 0000-1111 to all combinations of 0000-1111. These 256 simulations were run over- night. This took 64 minutes of CPU time plus 15 minutes of editting CPU time: a total of 79 minutes at $2000 per CPU hour, giving a nominal cost of $2650 just for accurately determining the longest path.

Note that there two processes are intimately related: if a fast method of doing individual SPICE runs is found then a convenient method of obtaining both the longest path *and* the stochastic simulation is found.

3.4.3 'Monte-Carlo' SPICE simulation

500 separate SPICE simulations were then run. Each one: generated random numbers for sampling the parameters from given Gaussians (truncated at 3-sigma); editted the nominal SPICE 'deck' to create a new deck for the run; and submitted that job for background execution. Each simulation was for 150nSec. The nominal deck is shown is Appendix F. Each run took about 30 seconds of CPU time but about 70 seconds of elapsed; time the total stochastic simulation for the leaf cell took just under 4 hours of CPU time and just under 10 hours elapsed time. The nominal cost was $2000 per CPU hour; this leaf cell thus cost $1333 to simulate. (This cost is for internal users, and thus somewhat under-estimates the total cost).

The next day (these large jobs run overnight) all the results were analysed by another set of programmes. They scanned each output file to find the time at which both the sum and the carry were ready (defined as both reaching 2.4 volts from 0 volts). At the same time, the value of the sum and carry separately at every 5nSec over the period 0-150nSec was stored in a histogram at every 0.1 volts within the range 0–5 volts.

Using nominal values of capacitance, resistance and threshold voltage, the nominal longest path is 100nSec. Using the extreme variations of these parameters yields times of 64 nSec and 155 nSec as the minimum and maximum times. However, when process parameters are correlated, extreme parameter values cannot be chosen independently, so the worst cases will actually produce rather misleading times – the extremes might not be attainable in reality.

3.5 The Significant Statistical Distributions

3.5.1 Probability Density Distribution

The nominal voltage waveform and the probability density functions for both the sum and the carry were compiled. The sum is shown in figure 3.1 on page 31. and the carry is shown in figure 3.2 on page 32. They are interesting because they show a multi-modal nature as time goes on, i.e., they do not remain strictly gaussian.

3.5.2 Accumulated Distribution Function

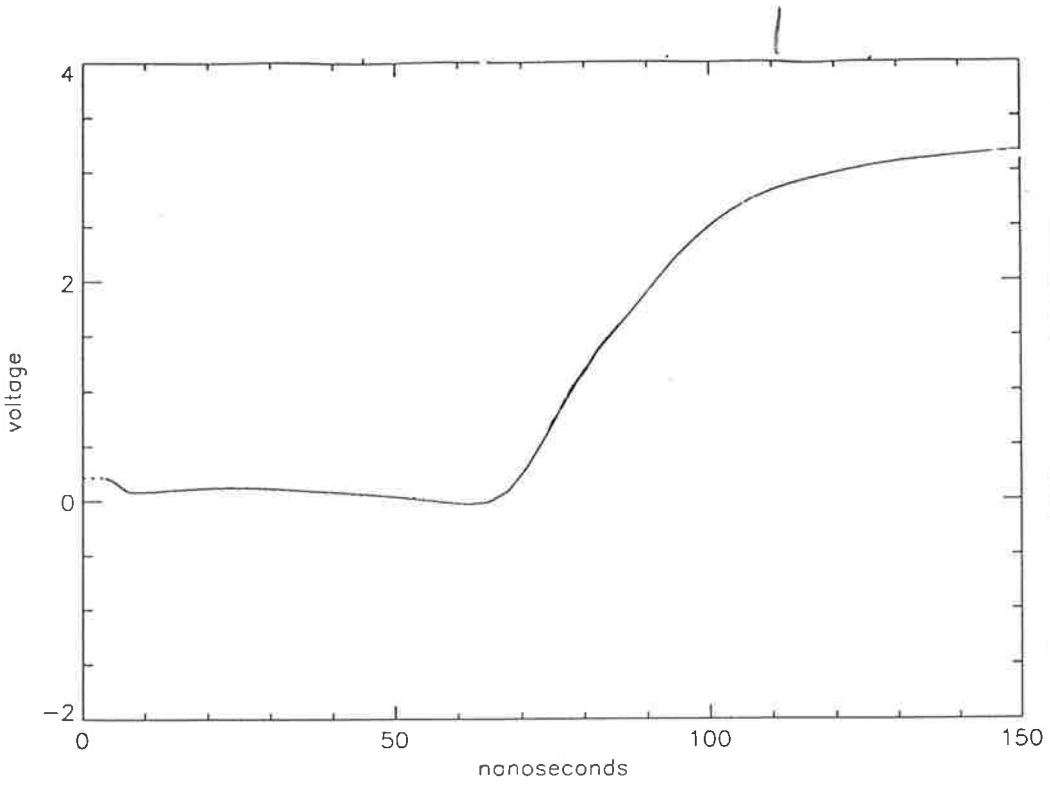
The first important result is the scatter histogram in figure 3.3 on page 33 showing the spread of finishing times. This shows that there is a spread of 'ready' times about the mean of 100 nSec, from 70 nSec up to 145 nSec. The distribution is skewed and sharply cut off at the fastest finish time.

From this is derived the second important result, the accumulated joint distribution function, shown above it. This is found by integrating the scatter histogram. Here *joint* means those cells for which both the sum and the carry have together reached 2.4 V. Thus it shows the proportion of leaf-cells that have completed operation at any time, and so it ultimately shows the yield of chips in terms of the design requirements.

3.5.3 Manufactured Operating Speed Distribution

The third important result is the spread of operating speeds of the chips. This is shown in figure 3.4 at the bottom of page 34.

Shown above it is the final important distribution derived from it by integration. It shows the proportion of circuits operating at least as fast as a given speed. Combined with the fabrication yield this can give the net yield of the design itself and hence is the most useful graph for deciding whether a re-design is necessary.



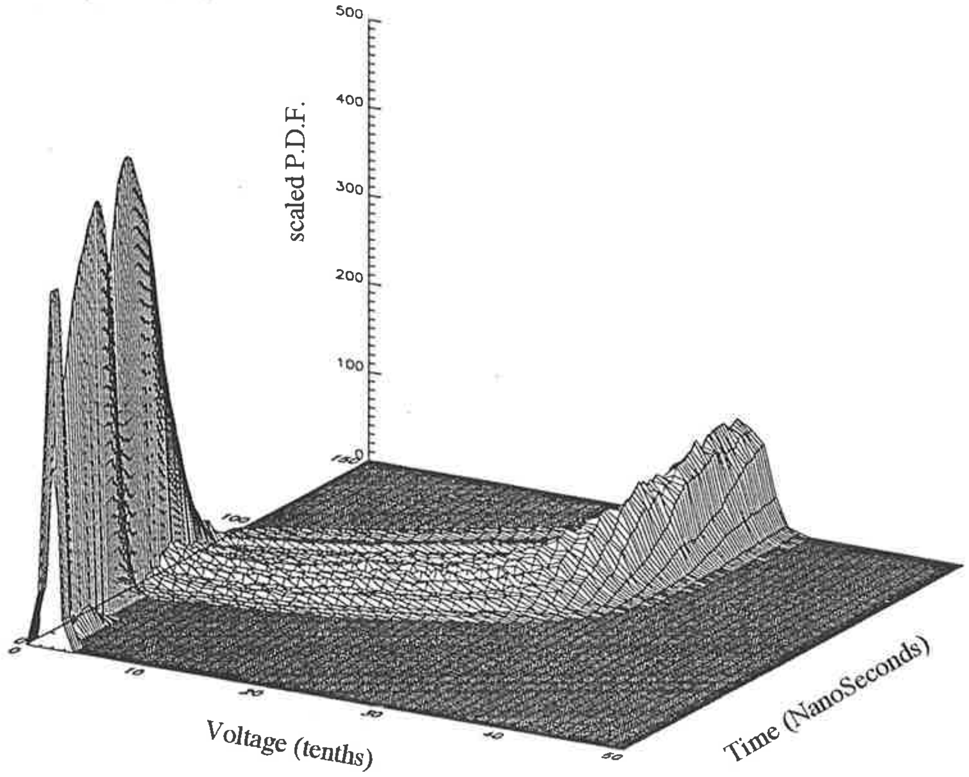
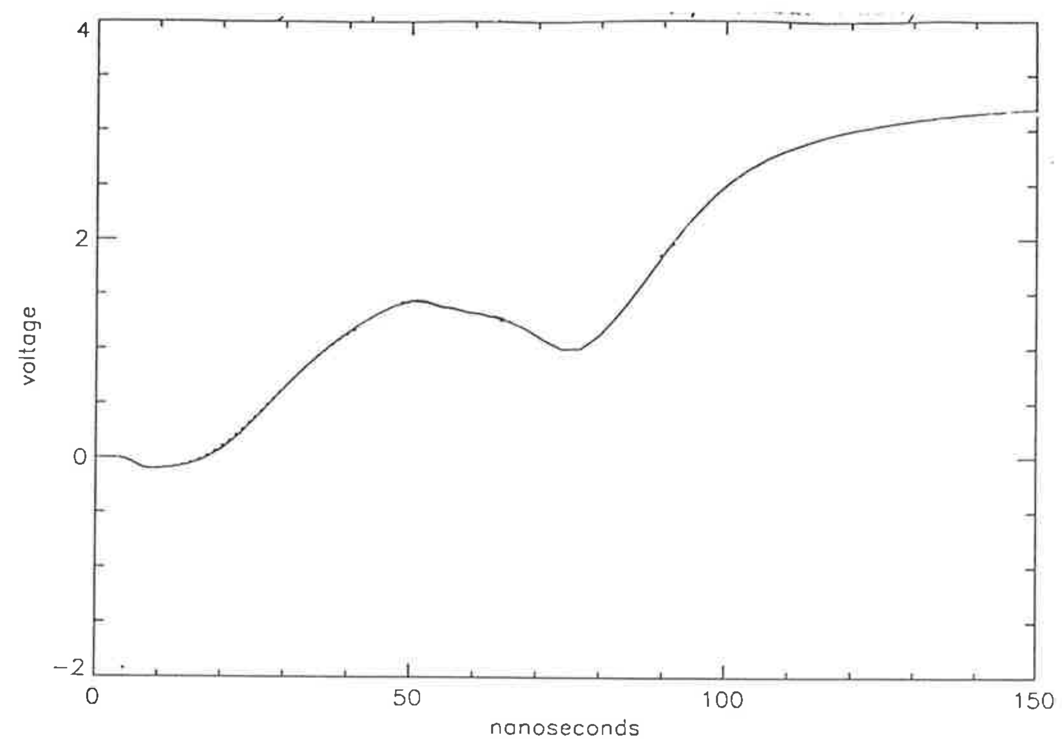


Figure 3.1: SUM: NOMINAL & PROBABILITY DENSITY FUNCTION



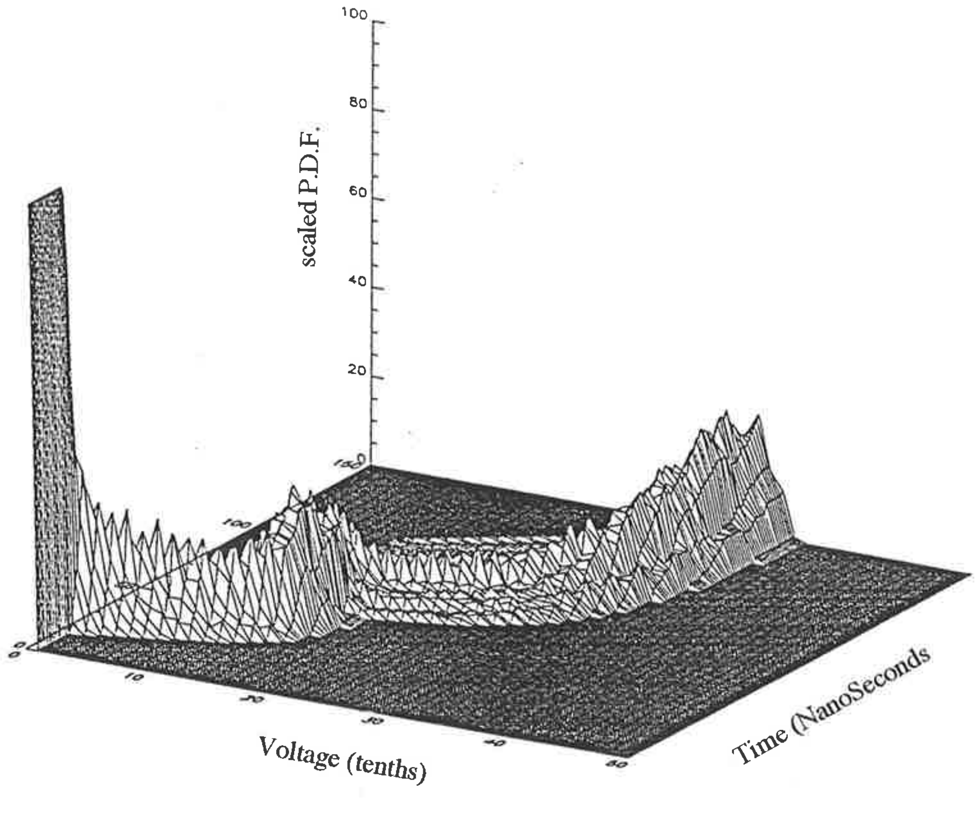
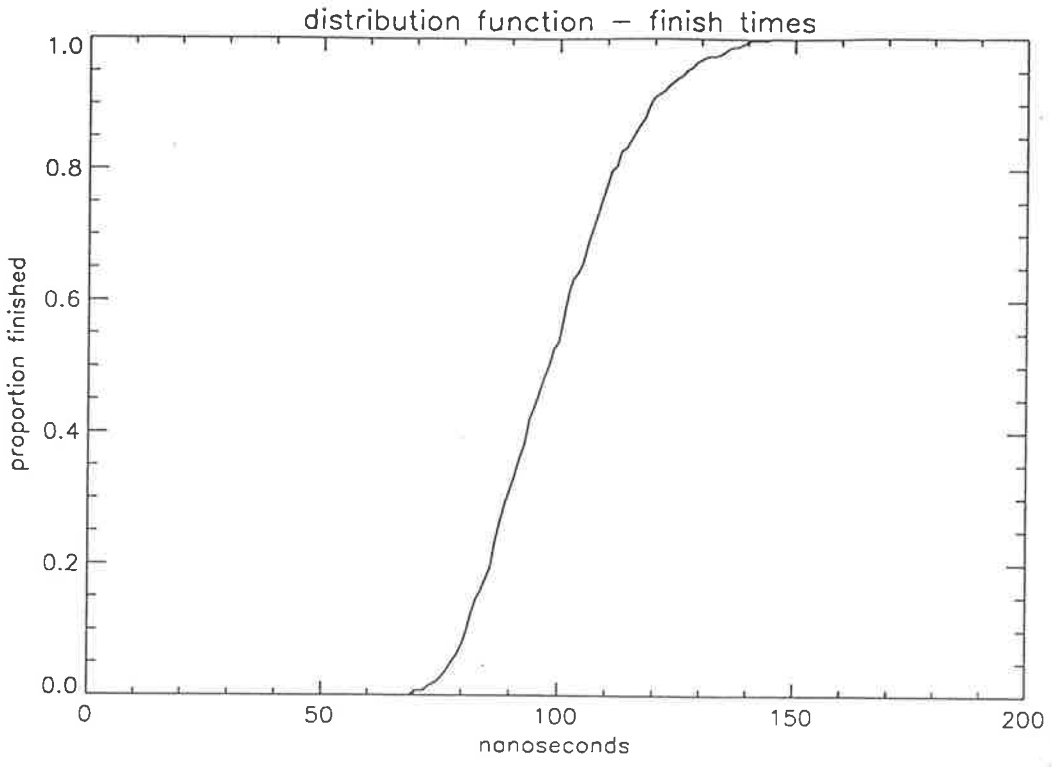


Figure 3.2: CARRY: NOMINAL & PROBABILITY DENSITY FUNCTION



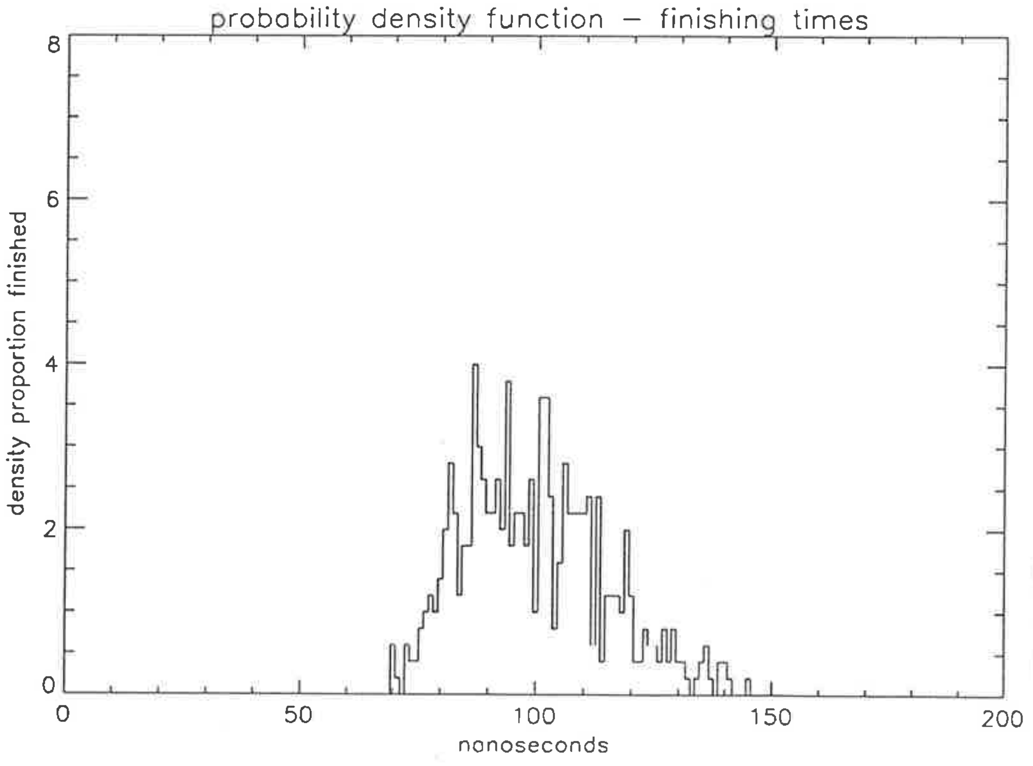
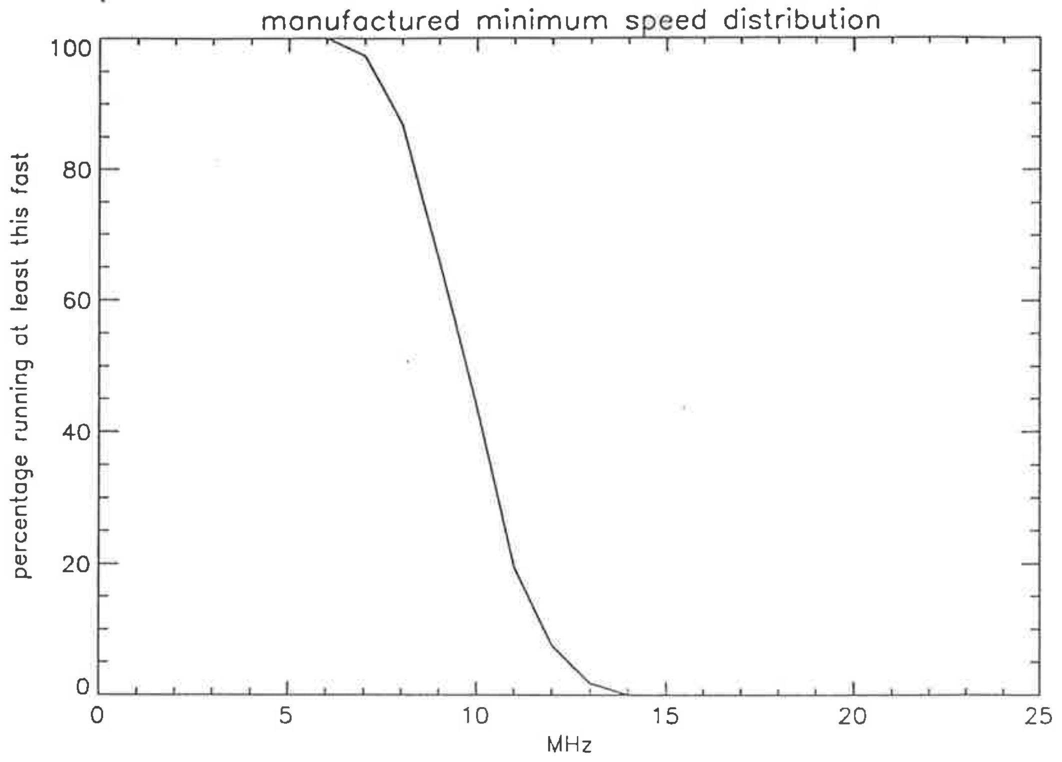


Figure 3.3: Spread of Finishing Times



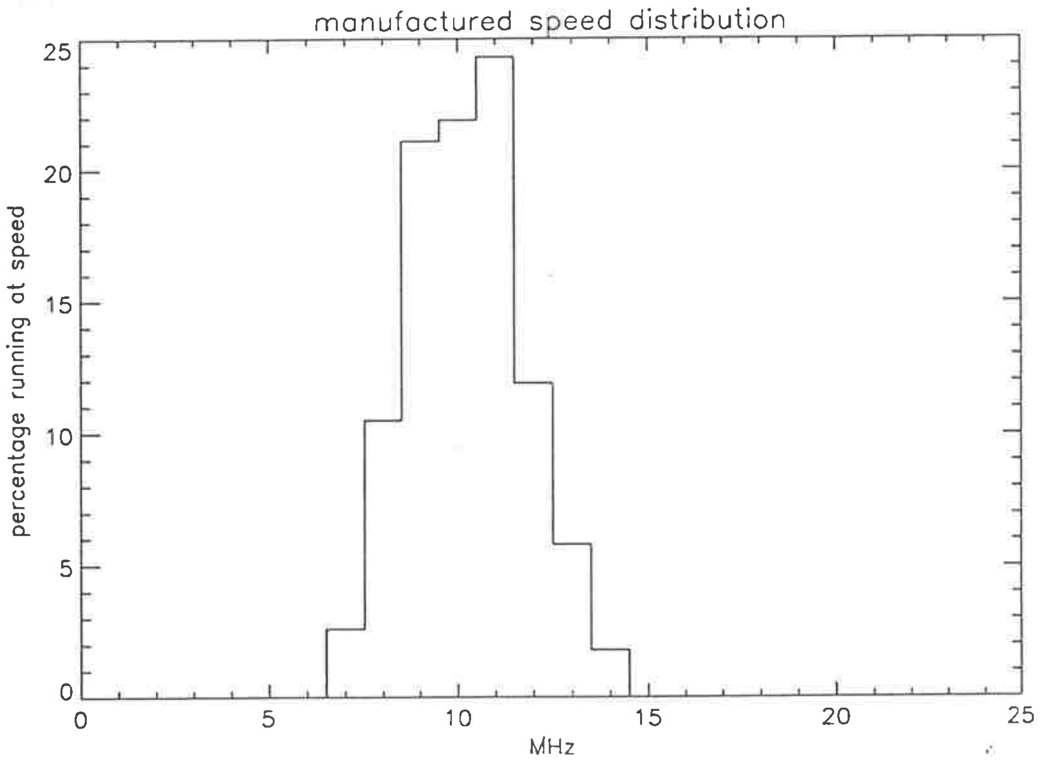


Figure 3.4: SPREAD OF MANUFACTURED MINIMUM OPERATING SPEEDS

3.6 Conclusions and Evaluation

Timing predictions for digital systems in VLSI circuits are made using various simulation programs. Statistical simulation is necessary when process parameters are not tightly defined; this is particularly important when predicting design yields, to help in deciding whether the circuit may be produced profitably at the designed speed.

These simulations make it clear that nominal design can be quite misleading for clocked systems. Only some form of stochastic design gives complete confidence in the results.

The only distribution of real interest is the accumulated joint distribution, which shows the time at which a given proportion of cells completed their task. From this, all decisions regarding re-design and fabrication can be made, since the speed and ultimately cost of each working chip may be obtained from it.

It is further suggested by the plots obtained that 500 runs is sufficient to get a very good timing characterisation. This is confirmed by other researchers – for example, Brayton finds that "Typically, at least 100 trials would be required to obtain a reasonable estimate for [the accumulated distribution]" [BS80] [BHSV81], others with more recent computing power [BS87] [SO86] [MSD86] [Kob82] [DCR82] quote between 300 and 3000 runs to get a good estimate. In addition, rather fortunately, "...the size of the sample does not depend on the number of parameters" [BHSV81].

The results of these simulations suggest that, for those intending fabrication at remote foundries, nominal design gives a dangerously false picture of the yield. Some form of stochastic design is necessary to give a measure of confidence in the results. It is already known that close to 500 simulations are necessary. Thus on the face of it stochastic simulation is a time-consuming business. The rest of this thesis sets out to deny that this is necessary.

Chapter 4

Existing Methods

In this chapter I review past and present methods of both stochastic and circuit simulation methods and identify areas needing further research.

4.1 Statistical Simulators

Design centering is the art of designing an IC to minimize the sensitivity of its fabricated performance[1] to the random disturbances affecting the manufacturing process. This is slightly different to the aims of this thesis, but there is substantial overlap.

Maly [Mal82] discusses a yield estimate algorithm which applies a learning procedure to a Monte Carlo approach. As sampling proceeds, information about the sample location is used to improve the efficiency of the yield estimation procedure.

Koblitz [Kob82] presents a graphical method for a design centering step, beginning with 100 Monte Carlo samples, which he regards as a small number. Past random samples of the Monte Carlo analysis are used in an updating process, as an assessment criterion for the next iteration. This way, the number of random samples can be reduced in each subsequent step; a reduction overall in the sample size of about 50 percent. He attaches great

[1]e.g., timing, power dissipation, or whatever is important for the application

importance to choosing an appropriate number of random samples in Monte Carlo analyses.

Downs, Cook and Rogers [DCR82] describe an approach to statistical design for large circuits, based on partitioning a system into subsystems. They expand the distribution function in an Edgeworth Series, and consider 3000 Monte Carlo simulations are required to give a 95% confidence interval on the yield, requiring 215 minutes on a PDP 11/40.

Canepa, Weber and Talley [CWT83] develop a standard worst-case design procedure on a short-channel NMOS III process. Most of the fabricated chips exceeded their performance specifications. Interestingly, they find that their program caught many fatal errors in the layout that could not be have been found by anything short of a chip-level circuit simulation.

Styblinski and Ruszczynski [SR83] apply an SA^2 algorithm to the yield optimisation problem. The yield estimator needs 300-1000 Monte Carlo trials. However, for circuit analysis that is excessive. They discuss parametric sampling, yield prediction and quasi-importance sampling, but the SA approach requires only a few extra samples each iteration.

Herr and Barnes [HB86] discuss a statistical design tool developed for production lines, using new MOS models of real processes. They stress the importance of considering parameter correlations from the outset.

Nassif, Strojwas and Director [NSD86] point out that yield prediction at each step of a process is prohibitively expensive and instead the IC design is usually verified under worst-case conditions. Parameter sets (threshold voltages, transconductances etc) are obtained from test structures but the correlation coefficients between device parameters are traditionally not taken into account. Device parameter distributions are estimated, extreme values for each parameter are chosen and combined to obtain extreme (worst) values of circuit performance. However, such combinations are unrealistic; the probability of simultaneous occurrence of these combinations is extremely low. Hence the results of such an analysis will always be much too pessimistic.

[2]Stochastic Approximation Technique

Correlations arise because different devices share processing steps, suggesting that worst-case analysis should be performed in terms of some lower level set of parameters, which will result in proper correlations between device model parameters. With the FABRICS tool they find that greater than 90 percent of the computational cost in the application of worst-case analysis methodology is the cost of circuit simulation. They conclude that full Monte Carlo simulation of complete ICs is not feasible, and worst-case analysis is an attractive alternative to verify an IC design.

Yang *et al* [YHC+86] start with the definition of parametric yield[3] and develop a statistical model using only interdie variations, since these are found to be much larger than those on one die. They show that current and capacitance of MOSFETS are primarily affected by: length reduction, width reduction, gate oxide thickness and flat band control, and create a quasi-physical circuit model with 9 parameters. Their *statistical parametric yield estimate* method, to within 5 percent accuracy, requires several hundred simulations to get a reasonable yield estimate, so they conclude that Monte Carlo has limited use for routine circuit designs.

Spoto, Coston and Hernandez [SCH86] explain that soon after variations in the fabrication process were found to cause parametric yield losses, the worst-worst case method was applied to the verification process and resulted in many overly designed products. Worst-worst[4] case was justified at the time because models were inaccurate and parametric data was either sparse or nonexistent. Their system integrates process, device, and circuit-characterisation functions. Using it, designers will no longer have to overdesign products to ensure their success.

Stein's method [Ste86] finds possible distributions for parameters which minimize the expected cost of producing design. Given simulation results for one parameter distribution, Stein gives a fast method which takes only a few extra samples, based on their demonstrated importance to the result, rather

[3]fraction of defect-free circuits meeting operating range performance specifications

[4]Worst-worst means extreme values without regard to parameter correlations

than redoing all the Monte Carlo simulations for a different distribution.

Styblinski and Opalski [SO86] develop a set of design centering tools within the FABRICS system. They note that the size of circuit that can be optimized is limited by the amount of time that can be devoted to optimisation and a major factor is the time taken by the SPICE circuit simulator. The overhead introduced by the optimization methods themselves is found to be negligible.

Alvarez *et al* [A$^+$88] discusses a new theoretical method for design centering, which uses information from parametric disturbance studies.

Mozumder, Strojwas and Bell [MSB88] concentrate on the statistical modelling of the process stage, while Director, Maly, Strojwas, Mozumder, and Bell, in a series of important papers [DMS88] [MSB88] [DMS88] [Str89] [Mal90] provide a consistent set of methods for increasing the fabrication yield.

4.2 Circuit Simulation

All the statistical techniques discussed above rely heavily on circuit simulation. Hon [Hon87] and Harrison *et al* [HNSB90] provide a good overview of circuit simulation tools within the CAD environment.

Newton [New79] recognised that while circuit simulation techniques can provide accurate waveform analysis of circuits of building-block complexity, the requirements of a circuit simulation become prohibitive as circuit size increases. Timing simulators can improve speed by two orders of magnitude while maintaining acceptable waveform accuracy by using node decoupling techniques and simplified lookup models for nonlinear devices, but the speed of timing simulation is insufficient for the analysis of large circuit blocks.

Yang and Chatterjee [YC82] recognise that statistical variation of parameters must be accounted for, and that parasitic capacitance and resistance are more dominant in the determination of transient response of integrated circuits as the feature sizes shrink and circuit density is increased. They present a comprehensive circuit simulation model for MOS short channel behaviour, including a model for the fringing capacitance due to finite gate thickness.

Vladimirescu and Pederson [VP82] describe *Classie*, a simulation program developed to narrow the speed performance gap between circuit simulators and timing verifiers.

Coughran *et al* [CGR83] report a damped-Newton method, proved effective in simulation, using splines for functional models of transistors.

Werner [Wer84], noting that the simulator is effectively the breadboard for system engineers doing custom IC design, reviews available switch-level, gate-level, functional-level, behavioural-level and mixed-mode simulators.

Newton and Sangiovanni-Vincentelli [NSV84] give an overview of circuit simulation programs, and methods used to improve the performance of conventional circuit simulators for the analysis of large circuits. They cover table look-up methods, direct methods and relaxation methods. For the direct sparse-matrix methods on which standard circuit simulators are based, a major drawback with the use of timing analysis is that tightly coupled feedback loops, or bidirectional circuit elements, can cause severe inaccuracies and even instability during the analysis.

Tsao and Chen [TC86a] describe MOTIS3, a *fast-timing*, multilevel mixed-mode simulator, with an automatic voltage step control scheme for optimising speed and accuracy. Circuits are first partitioned into subcircuits, and the simulator processes the subcircuit as one element, allowing event-driven techniques to be used for controlling the simulation mechanism and the circuit simulation techniques for circuit block evaluations. The *waveform relaxation* technique allows the compute time to be reduced at the expense of accuracy and vice versa. Up to 30,000 transistors have been simulated. They conclude that *fast-timing* simulation is about three orders of magnitude faster than conventional circuit simulation, and accurate to 5%.

Ruehli and Ditlow [RD93] discuss the SCALD timing verifier, using a calculus of 7 logic values to verify designs, and employing: the sparse tableau analysis method; modified nodal analysis method ; implicit integration methods; and sparse matrix techniques.

Benkoski and Strojwas [BS87] draw attention to the concept of solving for the time needed to cross a predefined voltage interval, as opposed to the formerly universal approach of computing the voltage reached at the end of a given time interval. This approach fits within event-driven frameworks, but is not general because it is only applied over a short time interval, rather than over a full waveform.

Bryant [Bry88] introduces switch-level simulation; Lee and Rennick [LR88] create ASIM, a compact IGFET model; Brocco [Bro88] review techniques of block construction for CMOS circuit models; and Cox, Burch and Epler [CBE86] discuss the circuit partitioning problem with an eye towards useful parallel processing methods.

4.2.1 Timing Verifiers

Hitchcock [Hit82] explains that timing verification consists of validating the length of path delays and checking the width of clock pulses. His program *TA*[5] generates standard deviations for the times so that a statistical timing design can be produced rather than a worst case approach. The accuracy of the answers is only as good as the accuracy of the delays calculated for each block, and assumes that the delays can be combined as for Gaussian distributions.

Jouppi [Jou87] shows that these block statistics of Hitchcock are assumed to be independent of the state of the circuit, so need to be compiled only once every design iteration. He describes TV, a MOS VLSI switch-level timing verifier, which analyses circuits with 40,000 transistors in under 30 minutes of VAX 11/780 CPU time. An interactive timing advisor provides incremental timing analysis. There are accuracy/speed tradeoffs involved in this approach. Jouppi, like Canepa previously, considers that analysis must

[5]Timing Adviser - for clocked, sequential machines

be performed on the entire design at once.

Banerjee [Ban88] presents a tutorial on timing verifiers within the context of design verification in the CAD environment.

4.2.2 Relaxation Methods

Hageman [Hag81] shows how systems of linear algebraic equations can be solved either by: direct methods, for systems of moderate size; or iterative methods, used primarily for solving large and complex problems for which, because of storage and arithmetic requirements, it would not be feasible or it would be less efficient to solve by a direct method.

Nevanlinna [Nev89] analytically attacks the question: how long to iterate?, because one of the main problems in iterative processes is to predict from the computed data when the true solution is already well approximated.

The *relaxation procedure* is an iterative method. Southwell [Sou43], the prime originator of this method, and also Shaw [SF53], both give a good explanation of this method, which was devised for the computation of stresses in frame-works and was shown to have application to other problems in engineering science. They assert that "discarding orthodox for relaxation methods, any problem that can be formulated can be solved".

Relaxation is a means whereby simultaneous equations may be solved, not exactly, but with steadily increasing approximation. Relaxation, or liquidation, of residuals is achieved in a number of steps, by applying the basic rule of relaxation at each step; the aim of every step is to change the value of the currently largest residual to zero. It can be applied to differential equations in their finite-difference formulation. Inaccuracies can be reduced by successively halving the time interval, errors do not accumulate at each step and high accuracy is not required.

The essential feature of the relaxation method: it fixes attention, not on the quantities whose values are required, but on the quantities whose values are given. Thus it has wider application than can be established by rigorous argument. It will still be possible that the solution is not unique; but this question is for physical intuition to decide.

Newton and Sangiovanni-Vincentelli [NSV84] concentrate on relaxation methods for the solution of the set of ordinary differential equations describing the circuit under analysis. The two most common methods used are the Gauss-Jacobi and the Gauss-Seidel. Relaxation methods are ideally suited to exploit any latency of the circuits, but are not guaranteed to converge.

The first successful application of relaxation methods to electrical-circuit analysis was in timing simulation. Timing simulators have proved successful when applied to constrained IC design methods, but have not been as successful in the custom-design environment. Since there is no way to guarantee accuracy for an arbitrary connection of MOSFET's unless at least two relaxation iterations are performed per time step, timing simulators have produced incorrect results in some situations.

An important assumption required by relaxation based electrical simulators is that a two-terminal capacitor be connected from each node of the circuit to the reference node (ground or supply).

Iterated timing analysis applies relaxation techniques at the nonlinear equation and its convergence properties are proven. The *waveform relaxation method* applies relaxation techniques at the differential equation level.

Newton and Sangiovanni-Vincentelli [NSV84] and Lelarasmee and Sangiovanni-Vincentelli [LSV82] both describe RELAX, a Waveform Relaxation MOS simulator. The algorithm used in RELAX assumes that the circuit consists of unidirectional subcircuits with no feedback paths. The method analyses a decomposed subcircuit for the entire simulation time interval, as opposed to only one time step, before proceeding to analyse another subcircuit. It is possible to show that this technique will always converge to the exact solution of the circuit differential equations provided that there is a grounded capacitor at every node in the circuit. Therefore the accuracy and reliability of this technique is guaranteed for most practical MOS circuits. If the subcircuits are processed according to the flow of signals in the circuit, the solution converges in just two iterations.

Experience simulating MOS digital circuits using RELAX2 shows that most MOS digital circuits without logic feedback loops converge in less than ten iterations. However, circuits with logic feedback loops may take many more iterations to converge, and the number of iterations required is proportional to the length of the simulation interval.

Iterated Timing Analysis, which can be derived from timing analysis, is accurate and fast, for large digital circuits.

The stability, consistency, and order of accuracy of the Implicit-Implicit-Explicit (IIE) method has been determined and the method is very promising for circuits with floating capacitors.

Newton and Sangiovanni-Vincentelli conclude that timing simulation algorithms are fast and rather accurate for the electrical simulation of MOS circuits with no tight feedback loops. Simulators using these methods provide accurate waveform information with up to two orders of magnitude speed improvement for large circuits. It is clear that relaxation-based algorithms for electrical simulation are well suited to the use of special-purpose hardware.

They conclude that the WR algorithm can handle floating capacitors and pass transistors satisfactorily, and RELAX can be at least one order of magnitude faster than SPICE2, for the same accuracy.

Dumlugol *et al* [DOCM87] discusses a mixed-mode switch electrical implementation of the *Segmented Waveform Relaxation Method*, an efficient waveform relaxation method for circuit-level simulation of large-scale digital MOS networks. The underlying idea is to decompose the large system of differential equations into subsystems, each of which is integrated independently on the given time interval, taking into account inputs from other subsystems from their state at the previous iteration. They note that the problem with timing simulation is that sometimes a wrong result can be produced without notice, so it has not found wide acceptance in the user community, except for certain classes of circuits.

Casinovi and Sangiovanni-Vincentelli [CSV88] discuss modern simulators, such as SPLICE or RELAX, which are based on relaxation methods because these methods are better suited to exploiting circuit latency and multirate behaviour. In all cases, their speed of convergence can be extremely slow, depending upon the numerical values of the elements of the circuit. They conclude that in the near future the application of relaxation methods to circuit simulation will remain confined to digital circuits in which no tight feedback loops are present.

Finally, Overhauser, Hajj and Hsu [OHH89] propose a scheme for automatic mixed-mode timing simulation. and Schneider [Sch91] gives a sufficient condition for the convergence of the Waveform Relaxation Method[6].

4.3 Fabrication Simulators

Garcia and Sriram [GS82] note that the main CAD tools for the design phase are the various types of simulators, but as design complexities shift from a circuit to a system level, other hierarchical simulators will become more important to the design process. In particular, the need for a set of models of the fabrication stage is recognised.

Mei and Dutton [MD83] write that with typical VLSI circuits consists of more than 100,000 transistors on a single chip less than 1 centimetre square with the minimum device feature size on the order of 1picometer or less, models for simulating both lithographic and etching steps are necessary and have been subject to intensive research, for example, the 2-D process simulation model, SUPRA, developed at Stanford University.

Lightner [Lig87] notes that wide use of workstations and personal computer based CAD systems is increasing the demand for efficient and accurate multi-level simulators, not only spanning the logic and behaviour levels but including the circuit and switch levels as well.

[6]also referred to as the Picard-Lindelof iteration

4.4 Simulation Acceleration

4.4.1 Simulated Annealing

Kuh and Ohtsuki [KO90] criticise the 'sea-of-gates' design style which calls for efficient algorithms which deal with hundreds of thousands of gates. Previously useful algorithms based on simulated annealing or other random methods have become unusable: even for them the computation time is unbearable in spite of the growing accessibility of faster computers.

4.4.2 Software Speedup

Saleh *et al* [SGC$^+$89] know that time-domain transient-analysis circuit simulation is a very time-consuming and numerically intensive application, especially in the case of VLSI circuits. To improve performance without sacrificing accuracy, a variety of parallel processing algorithms have been investigated. Both standard direct methods and relaxation-based approaches such as waveform relaxation, iterated timing analysis and waveform-relaxation-Newton are explained and their problems explored. They consider parallel direct methods, model evaluation, sparse system solver, iterated timing analysis and parallel relaxation methods. In particular, the forms of parallelism available within the direct method approach, used in programs such as SPICE2 and SLATE, and within the relaxation-based approaches, such as wave-form relaxation, iterated timing analysis, and wave-form-relaxation-Newton, are described.

For example, parallel circuit simulation programs based on nonlinear relaxation have been developed for the BBN Butterfly, and the Sequent multiprocess. A number of parallel simulators using direct methods have also been developed primarily related to general-purpose multiprocessors with a shared-memory architecture having a limited number of processors.

They conclude that the convergence speed of relaxation methods depends on the degree of coupling between the equations in the system and the order in which the equations are processed. If some equations are tightly coupled convergence can be very slow.

4.4.3 Hardware Speedup

The evolution of technology provides the opportunity of forming large processor ensembles working closely together. Instead of building a small number of extremely powerful supercomputers it is most promising to develop methods for distributing large jobs over many computers of the usual size. These are manufactured in large numbers and therefore are cheaper. Multiprocessors come in various sizes, ranging form two processors connected by a simple serial link to computers with tens of thousands of processors connected by a complex communications network.

Arden and Ginosar [AG82] explain that, historically, the large processing rates characteristic of supercomputers have been produced by two different approaches to concurrency. In the first, or pipeline approach, operations are divided into small roughly equal duration units which can be executed in parallel using very high performance circuits. In the second approach concurrency is more macroscopic. A relatively large number of simpler processors can execute simultaneously on larger parts of the algorithm being executed. They argue that the former has been more successful, but that the latter holds more promise for future general computation since the simpler processors provided a modularization that is compatible with anticipated VLSI circuit capabilities, and the need to accommodate relatively large computations "on chip". They discuss a number of examples:the distributed, extensible, MP/C is a MIMD machine with a dedicated-path network; the Colombia Homogeneous Parallel Processor is another proposed MIMD; the Minerva, a single bus MIMD, and the Dynamic Computer both have a similar organisation to the MP/C; the Cm* and the X-Tree both have processor-to-memory communication systems; the Divided and Conquer computer (DAC) is a multicomputer with a binary tree interconnection network; and the binary tree machine proposed by Bentley and Kung is similar to the DAC.

Levendel, Menon and Patel [LMP82] present the architecture of a special-purpose computer for logic simulation using a distributed processing network based on an interconnection of low cost microcomputers. The circuit to be simulated is partitioned into subcircuits and each subcircuit is simulated in a separate microcomputer. Thus, several microcomputers can be simultaneously simulating several elements activated by parallel signals. A combination of the cross-point matrix and parallel bus can be used to simulate circuits containing both simple and functional elements. The speed/performance ratio of the simulator is expected to be greater than two orders of magnitude compared to traditional simulation methods implemented on general-purpose computers.

Bondurant [BKB82] discusses the parallel-pipeline array processor architecture of the Honeywell Array Processor, designed for military applications.

The Yorktown Simulation Engine [DKP83] is a programmable machine which can simulate up to 1 million gates at a speed of over 2 billion gate simulations per second. It is mentioned here because custom transistor-level FET designs, including pass transistor logic and charge sharing, can be simulated by converting the source network description into a functionally equivalent gate network.

Fromm *et al* [FHJ+83] considers the EGPA Pyramid, a hierarchical multiprocessor operating system, and discusses interconnection structures such as time shared or common bus systems, crossbar switch matrix, and multiport memory.

Gottlieb *et al* [GGK+83] present the NYU Ultracomputer, a shared-memory MIMD parallel machine composed of thousands of autonomous processing elements. The design is a general purpose MIMD machine with a message switching Omega-network. Their simulations lead them to reject SIMD machines in favour of the MIMD model for certain types of simulation.

Ambler [Amb85] intends to exploit the inherent parallelism in circuits by modelling transient operation node-for-node on an array of cooperating microprocessors - transputers - with 10-100 processors under control of a mainframe computer. This method exploits the circuit's latency - only 10% of its logic is active at any moment. For example, a circuit with 1 million nodes could be partitioned into 400 node modules and the modules simulated in series on a 400-processor array.

Dettmer [Det86] discusses a transputer design used for graphics. Each IMS B003 has four T414 32-bit transputers with 256 kbytes of DRAM per transputer. The transputers are organised in a particularly simple way. There is one master transputer and 39 slave transputers which receive work from the master.

May and Fuge [MF86a] discuss the T800, developed as part of the European Esprit parallel-computer-architecture project.

Gabriel [Gab86] introduces and discusses a wide range of massively parallel computers, their communication networks and memory arrangements, such as the Goodyear Massively Parallel Processor, built in 1979. This machine is a 128 x 128 grid of 64-bit, 100-nsec processors, each with 1024 bits of RAM, and packaged eight per chip.

The major problem to solve in building a massively parallel computer is how to interconnect a very large number of processors and memory modules. There is a class of connection strategies whose hardware requirements grow as $log(n)$; this family uses a network known as the omega or butterfly network, a member of a class of networks called shuffle-exchange networks. This is the communication architecture used by the Bolt, Beranek and Newman Butterfly computer.

Another application of the omega network is the hypercube or Boolean n-cube. It takes at most $log(n)$ time to transmit a message through an n-dimensional hypercube. The Connection Machine, a SIMD computer comprising 65,536 (2^{16}) processors connected as a Boolean n-cube, uses the hypercube connection scheme.

The NON-VON computer is discussed. There are two categories of processors in the NON-VON, the small processing element (SPE) and the large processing element (LPE). SPE's operate in SIMD mode under the control of the LPE's. NON-VON can support SIMD, MIMD (multiple-instruction multiple-data), and MSIMD (multiple SIMD) operations. Algorithms such as relaxation are naturally performed on such processors because the structure of the processor matches the structure of the problem and the problem solution Gabriel concludes that for $n > 10,000$ the cost of the crossbar switch is prohibitive and its size unmanageable.

Charlesworth [CG86] discusses supercomputing with replicated VLSI, and explores the operations of: the CMU Systolic Array Computer; Intel's Personal SuperComputer (iPSC); the 16-processor C.mmp; and the FPS-164/MAX matrix supercomputer.

Dumlugol *et al* [DOCM87] consider hardware acceleration of *switch electrical waveform relaxation* methods on parallel computers where a number of processors communicate over a parallel bus. Such algorithms are well suited for parallel computation since several subnetworks can be simulated concurrently on different processors over time intervals containing multiple time points. The Gauss-Seidel WR algorithm, together with the timepoint pipelining method, allow substantial acceleration on parallel computers, but acceleration drops for more than about 16 processors. They discuss an advanced timing-simulation method which does not suffer from the accuracy and stability problems of previous timing simulators and which substantially increases the algorithmic parallelism. Speed gains over the direct method of more than an order of magnitude are obtained with SWRM.

Odent, Claesen and De Man [OCD89] discuss the problem of feedback loops in parallel relaxation-based simulators.

Soule and Blank [SB88] look at the subset of logic simulation on general purpose machines, while Peipho and Wu [PW89] offer a good comparison of RISC architectures.

August *et al* [ABHS89] devote a paper entirely to the Cray X-MP.

Lopez and Valimohamed [LV90] introduce the concept of a software environment for developing engineering application systems for multiprocessor hardware (MIMD). Distributed-memory, or loosely coupled systems, are networks of computing elements each having its own memory. No memory is globally accessible. Communication and data transfer in such systems are accomplished by "message passing", e.g., the Hypercube and the Intel iPSC. They conclude that a large percentage of software research in parallel computing has been directed at defining new forms of old algorithms that are more suited to a specific type of MIMD computer.

Ortiz [OPP91] describes the Connection Machine, a hypercube-connected network of clusters of 32 processors, each of which is connected to 64K of random access memory, a 32-bit floating point processor (FPP) and a FPP chip. Machine configurations range from 8K to 64K processors. Message-passing between arbitrary processors in handled by a hardware router. Stiller [Sti91] looks at the next model: the Connection Machine-2. The CM-2 has a fine-grained SIMD architecture with 64K processing elements controlled by a standard front end interface. Each processing element is bit-serial, has between 8K and 128K bytes of bit-addressable RAM, and is connected in a binary 16-cube.

Lin and Wu [LW92] also explore the n-dimensional hypercube (binary n-cube): a highly concurrent loosely coupled multiprocessor consisting of 2^n identical processors. Each processor of node has its own local memory. They conclude that the meshes of processors are very suitable for implementing many iterative or recursive algorithms in parallel.

4.4.4 Parallel Systems

Swartzlander and Gilbert [SG82] examine three of the classical design options: high speed monoprocessors, array processors and distributed processor. A common design theme is that the most efficient algorithms are all multiple instruction stream, multiple data stream (MIMD) devices. They conclude that the crossbar arrangement has been used only in those instances in which maximum interconnection flexibility is required and where there are a relatively small number of nodes. Parallel and pipeline processors lack the required flexibility for many supersystems applications, so attention has been focused on distributed networks. The later approach appears most desirable for supersystems, but requires improved interconnection networks.

Teja [Tej85] also considers architectures that distribute the processing to multiple fairly-independent CPUs. Somewhere between a simple reassignment of routine tasks to slave processors and true parallel processing is a class of high-powered computers whose architectures are best described as distributed. Sequent Computer Systems' Balance Computer, for example, allows system configurations with as few as 2 or as many as 12 processors.

Agrawal and Jagadish [AJ88] examine the important issue of how to partition the circuit for simulation on a class of parallel architectures.

Lewis [Lew88] presents the design of a hardware engine for timing simulation using a nodal formulation and the forward Euler integration algorithm to solve the differential equations that model the circuit, and a multiprocessor structure with a high-band width interconnection network to support parallel simulation. This *iterated timing analysis* method exploits circuit latency and uses multirate integration. The speed of this accelerator is extremely high, allowing the simulation of circuits with over 100 000 transistors in under one second per simulated clock cycle. Each node has a capacitor with one terminal connected to ground, and no other capacitances are allowed. He only simulates accurately enough to avoid the problems of switch-level simulation and obtain correct logical results and reasonable timing accuracy.

The circuit is partitioned into a number of subcircuits, where each subcircuit can be an arbitrary collection of nodes and the devices connected to them. Each processor is responsible for simulating the nodes in one or more subcircuits. The simulation of a subcircuit may require knowledge of voltages of nodes in other sub circuits, possibly on other processors.

AWSIM uses an 11 bit representation because the intended application does not require high accuracy. It has been used to simulate designs containing up to 13,000 transistors. The net speedup due to multiprocessing is about 9 times, for 32 processors.

Lewis admits that the principal disadvantage of Awsim is its poor modelling of parasitics and inability to handle floating capacitances. Furthermore, many timing simulators are more accurate than Awsim because of better algorithms and device modelling. Finally, they run on general-purpose hardware that can be used for many different tasks.

However, one of the surprising aspects of this machine is the use of a forward Euler integration algorithm, long discarded for most software implementations because the poor stability of this algorithm results in a small time step. As a result, an algorithm which appears to be obsolete for software implementations results in an inexpensive hardware implementation with good performance. Once again, notably, the ability to perform a timing simulation on the entire chip was found to be essential for design verification.

Other multiprocessor simulators use an event-driven message passing scheme include MSPLICE, a multiprocessor implementation of ITA using shared memory, and Event-EMU of Ackland and Clark [AC89].

Renterghem [Ren89] describes in detail the transputer of INMOS, designed to facilitate loosely coupled multiprocessor system. It has a multitasking kernel and schedular embedded in the microprocessor's microcode. A transputer can pass messages faster to surrounding transputers than to transputers that are not directly connected to it (messages have to be routed through one or more other transputers). In a transputer system, there is no bandwidth saturation as the system size increases, no capacitive load penalty as more transputers area added and no communication bus contention. Transputer systems normally act as coarse grain size computers.

He discusses various forms of applicable parallelism, including: geometric parallelism; algorithmic parallelism; data parallelism; and hybrid forms of parallelism.

Hayes and Mudge [HM89] review a range of multiprocessors, all having a few dozen processors connected to a shared memory over a common high-speed bus. Examples are the Sequent Balance and the Encore Multimax. For the BBN Butterfly and the RP3, a key feature is the omega-type multistage interconnection network that connects the processors to the shared memory.

The earliest study of hypercube computers was published by Squire and Palais of the University of Michigan in 1963. Their stated goal was to design a computer "where the emphasis is on the programmability of highly parallel numerical computations, *with hardware cost a secondary consideration*[7]".

With the advent of the single-chip microprocessor in the early 1970's several other proposals for microprocessor-based hypercubes were made. IMS Associates announced a 256-node commercial hypercube based on the Intel8080 micro-processor, but it was never produced.

The Colombia Homogeneous Parallel Processor, which would have contained up to a million processors, was proposed in 1983 and not built, but the 64-Node Cosmic Cube was built in 1983. In 1985: Intel delivered the first production hypercube, the iPSC; NCUBE Corporation produced the NCUBE/ten; and System 14/n came out from Ametek. Other parallel computers were the Caltech/JPL Mark 3, the Floating Point Systems T Series, and the Intel iPSC/2. The Connection Machine series manufactured by Thinking Machines Corporation employs up to 2^{16} simple processing nodes.

They find with their CL algorithm, the speedup is reasonable for up to 16 processors, and that little is gained by increasing the number of processors beyond that. Hayes and Mudge conclude that high hardware cost was clearly a major reasons why these early hypercube designs were never implemented.

Duncan [Dun90], Hennessy and Patterson [HP90] and Skillicorn [Ski91] all provide useful surveys and models of parallel architectures and quantitative and computational methodologies.

Vaughan [Vau92] introduces the basic idea underlying the multiple-virtual-ring (MVR) message-passing system, with its protocol of taking a divide-and-conquer approach to information-distribution policies in distributed systems, and investigate networks of up to 20 homogenous processors. He concludes that the implementation of a divide-and- conquer philos-

[7]my emphasis

ophy does not imply an increase in the overhead of information distribution, and that much work remains to be done on the effect of system size, interprocessor distance and distance between subsystems in networks of more than 100 processors.

Steven *et al* [SAFT92] studies the iHARP processor, which fetches a 128-bit long instruction word from an instruction cache every processor cycle. These are often called VLIW (very long instruction word) architectures.

Bogineni and Dowd [BD92] invent an optical interconnect system, with which they hope to ease the backplane interconnection problem; while Clark, McColm and Stark [CMS92] address some of the related problems facing certain architectures.

4.5 Conclusions

Important issues, including some unaddressed problems, include:

Form of the Distribution This has not yet been discovered, but this fact is addressed in Part III of this thesis.

Number of Monte Carlo Simulations Estimates of the number needed average around 500 Monte Carlo runs. Thus the circuit simulation task is the overwhelming bottleneck. This is also addressed in Part III.

High Accuracy High accuracy is generally considered necessary, with less that 1 percent often quoted.

Modelling Strategies There are many methods, dominated by relaxation, in software, and by distributed MIMD architectures, in hardware.

Conclusions normally reached are: that the convergence speed of relaxation methods depends on the degree of coupling between the equations in the system and the order in which the equations are processed. If some equations are tightly coupled convergence can be very slow. Against this, it has to be borne in mind that the relaxation method was invented at a time when there was only very rudimentary computing machinery, and that, in any case, the MIMD architecture has some real problems as the scale increases.

Whole-Chip Simulation There seems to be a recurring desire to simulate large parts or all the chip circuit at once. Although this may in cases be due to irregular design methods, it seems to be a genuine requirement. However, it is none-the-less alarming to find Jouppi [Jou87] noting that:

'Recently, integrated circuits themselves have become so large that their timing is not easily understood by their designers'

in the context of making the point that

'simulations must be pieced together by making assumptions about the interaction of design blocks. These assumptions are complex and prone to error. To eliminate these assumptions, analysis must be performed on the entire design at once.'

Migration to Workstations All the tools reviewed here have been developed and are available on mainframe computers in large production companies or universities. Large companies have quite different aims from small designer groups, who desire remote fabrication of limited runs.

Werner [Wer84] notes that workstation-makers are investigating the use of commercially available array processors to speed up logic simulation.

None-the-less, none of the stochastic methods have migrated but since the overheads are small that should present no problems.

In this thesis, heuristic methods will be preferred, rather than a full design centering implementation, to get a feel for how well designs will manufacture at remote sites. Suitable methods for this are addressed in Part IV.

Part III
New Ideas

"I speak without exaggeration when I say that I have constructed 3,000 different theories in connection with the electric light, each one of them reasonable and apparently likely to be true. Yet in two cases only did my experiments prove the truth of my theory."

THOMAS EDISON

Chapter 5

Concatenated Structures

Processing-parameter variations occurring during fabrication of VLSI circuits cause a statistical spread in circuit-element operating times.

This spread is easily found for single leaf cells, but not for concatenated structures (adders, etc.) due to long time and large size constraints on the simulator. Here I invent an accurate method of obtaining the spread of concatenated times directly from the spread of the operating times of just the first two cells of such a concatenated structure.

5.1 Introduction

The effects of fabrication-parameter variations on the operating time of a single leaf cell have been investigated in a previous chapter. From these results it might be argued that the effect is small and that the spread of times is not large enough to be of practical concern. It will soon be seen that this is definitely not so. The reason that the spread becomes very significant in real circuits is that the time of operation of most circuits depends on subsystems which are made by simple concatenation of a leaf-cell; ripple adders and parity generators are examples of this. Thus an initial small spread of finishing times for a single cell can become greatly magnified in the final concatenated structure, which might consist of from 8 to 24 cells or more.

In these cases the statistical behaviour – that is, the spread in the finishing times of the circuit element – under conditions of uncertainty in process parameters becomes a much larger and less general problem. It is larger because many more cells are involved in the circuit simulation. It is less general because this very size prohibits the application of the single-cell methods without some adaptation.

Consider a simple shift register, or actually a sequence of cascaded or concatenated inverters (like a ring oscillator, perhaps). A signal allowed to 'pulse' into the first cell (via a clocked line) has a given distribution of times. However the input to the next cell is quite different in general from this initial signal and so the next cell has a different delay and a different distribution function. If the output of each cell is input to the next then a formidable problem (for SPICE) is faced with a single 16-bit register. However if only the first and second cells need stochastic modelling then the distribution function of the overall register is obtained with huge savings in both money and time. Thus this algorithmic approach to statistical modelling of large registers reduces it to practicality.

Thus there appeared a need to obtain structural timing statistics from simple cell statistics, as an inexpensive way of determining design yield, rather than having to stimulate an entire structure consisting of many identical cells.

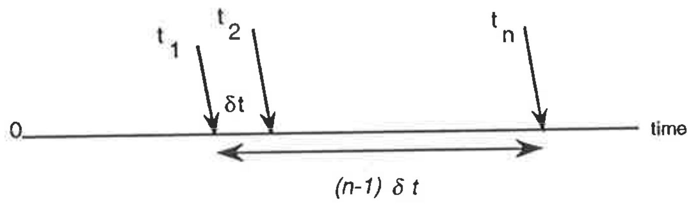
5.2 The Heuristic Stage

While developing another application using cascaded inverters, the results produced by 150 stochastic SPICE simulations of the cascaded inverter timing showed that, after the first cell (which is a special case) the mean finishing time of each stage increased linearly with the stage.

5.3 An Hypothesis

These results suggested that the distribution for n stages might be obtained with some accuracy by suitably combining the results from the first stage alone with the results from the first two stages considered jointly.

In what will be referred to here as the TIME method, it is proposed that for each finishing time resulting from a Monte-Carlo simulation, the finishing time of the nth stage is predicted by linear extrapolation of the time differences between the second and first cells - i.e.,



$$t_n = t_1 + (n-1)\delta t$$

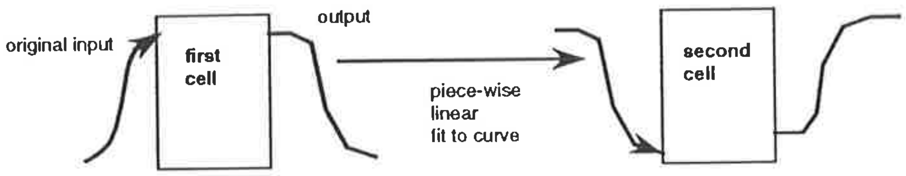
The cumulative distribution is then formed directly from the histogram of these predicted times.

This hypothesis was strengthened by a systematic experiment involving a cell cascaded into an 8-bit adder. The experiment is described in detail below.

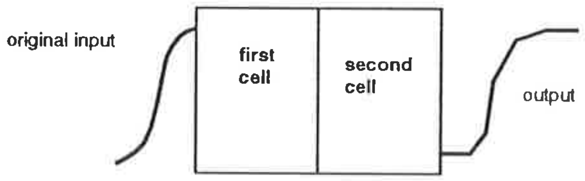
5.4 Testing the Hypothesis

This TIME method was tested in a systematic study using an adder cell adapted from a multi-project chip design, described in Appendix A.

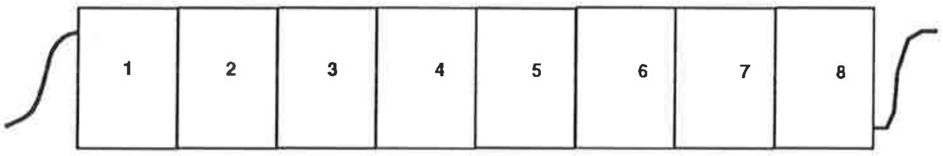
The actual spread for an 8-stage adder was produced by running 200 SPICE simulations, sampling the process parameters from gaussian distributions. The spread at each stage is shown in figure 5.1 on page 64. The output voltages from each stage were fitted by a piece-wise linear fit and this was used as the input to the next stage in the adder, thus:



when a further SPICE simulation was done on the next stage. This then becomes the equivalent of:



This procedure avoids making the simulator twice as large as for a single cell, since two cells are not being simulated at once, but only one cell (the first) during the first stage, and then only one cell again (the second) during the second stage, when the output from the first stage is fed in as input to the second cell. This elementary process was repeated for the whole 8 stages, ultimately producing



The spread of process parameter values used in the simulations cover three standard deviations (99% of all samples) and these spreads have already been summarised in the table on page 28. The values roughly approximate those used for making timing predictions for AUSMPC-5/82; however the spreads were for simulation purposes only and do not correspond to any particular process.

5.5 Experimental Results

The extreme finishing times found below demonstrate quite clearly that the spread becomes very significant as an initial small spread of finishing times in the single-cell is greatly magnified in the 8-cell final structure.

STAGE	MINIMUM TIME *nSec*	MAXIMUM TIME *nSec*
1	45	98
2	88	222
3	130	296
4	173	421
5	215	477
6	256	620
7	298	659
8	340	819

Interestingly, it is found that it is not the variance, skewness and kurtosis themselves that increase linearly with each stage. Rather, the quantities that increase linearly with each stage, and are plotted later, are the related mean linear measures: (compare definitions on page 89)

$$\mu = \frac{1}{N}\sqrt[1]{\sum_{i=1}^{N} t_i}$$

$$\frac{1}{N}\sqrt[2]{\sum_{i=1}^{N}(t_i-\mu)^2}$$

$$\frac{1}{N}\sqrt[3]{\sum_{i=1}^{N}(t_i-\mu)^3}$$

$$\frac{1}{N}\sqrt[4]{\sum_{i=1}^{N}(t_i-\mu)^4}.$$

Shown next is a table of these related mean linear measures for the 8-stage cell experiment, along with the predictions of each of these parameters made by the TIME method described above.

STAGE	MEAN	VARIANCE	SKEWNESS	KURTOSIS
Actual				
1	64.939	0.840	-0.373	0.302
2	122.108	1.548	-0.707	0.567
3	178.993	2.288	-1.052	0.838
4	236.118	3.009	-1.389	1.105
5	293.250	3.731	-1.726	1.372
6	350.403	4.453	-2.064	1.639
7	407.571	5.175	-2.401	1.907
8	464.766	5.898	-2.738	2.174
Predicted				
8	465.119	5.816	-2.705	2.156

The results in figure 5.1 on page 64 show the timing histogram for each of the 8 stages (cells of the ripple-carry adder) from zero to 800 nanoseconds. The spread increases at each stage and the mean time also increases linearly with each stage. Figure 5.2 on page 65 shows the overall statistics of this computer experiment: the top plot showing how the statistical moments develop with each of the 8 stages of the adder; the middle plot showing the cumulative distribution for each stage, from 0% cells finished operation up to 100% done over 0-800nSec; and the bottom plot showing the proportion of chips operating at least at a given speed.

Figure 5.3 on page 66 shows the comparison of the actual and predicted distributions. Plotted on top of one another on the same graph, and almost indistinguishable because they match so well, are the predicted and actual cumulative distributions.

5.6 Evaluation and Conclusion

The TIME method is found to be astonishingly accurate. This is confirmed using a χ^2 test by fitting the predicted probability density function to the actual one, using this method. It is found that with 23 degrees of freedom, $\chi^2 = 1.5$, which means that the probability of a worse fit than this one (by chance alone) is effectively 100%. Thus the fit obtained here can scarcely be improved upon.

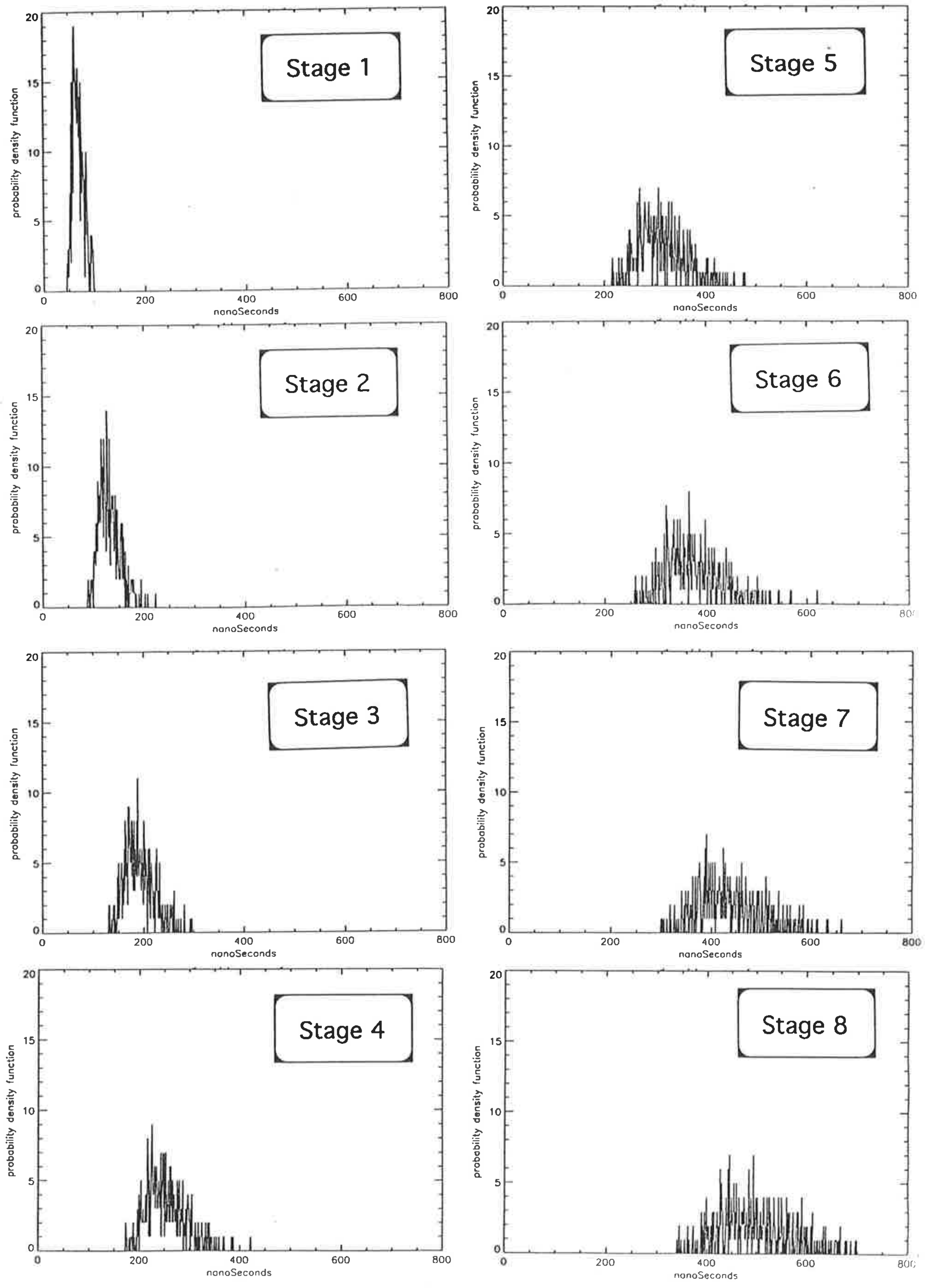


Figure 5.1: 8-Stage Adder Cell Timing Spread

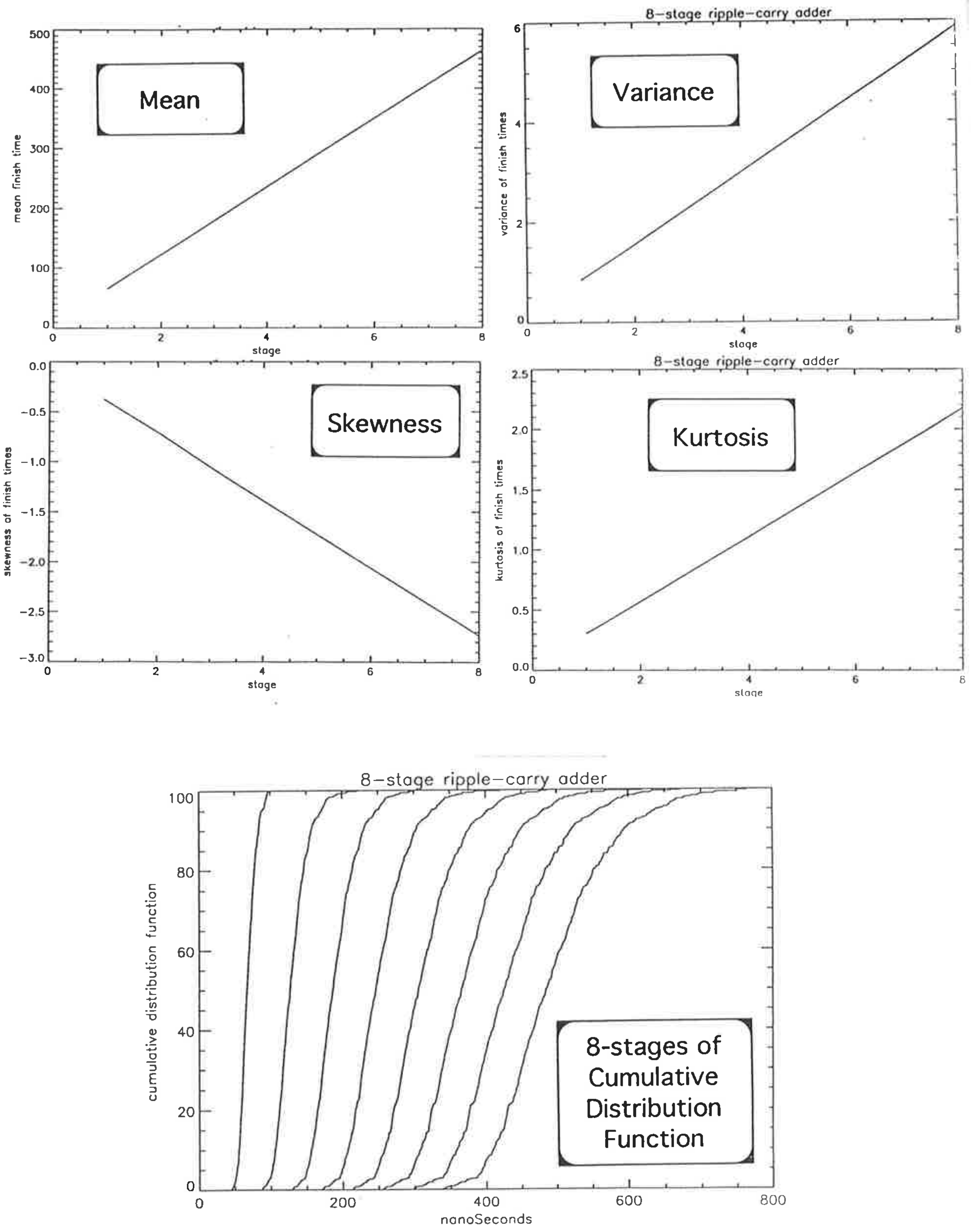


Figure 5.2: 8-Stage Adder Cell - Overall Statistics

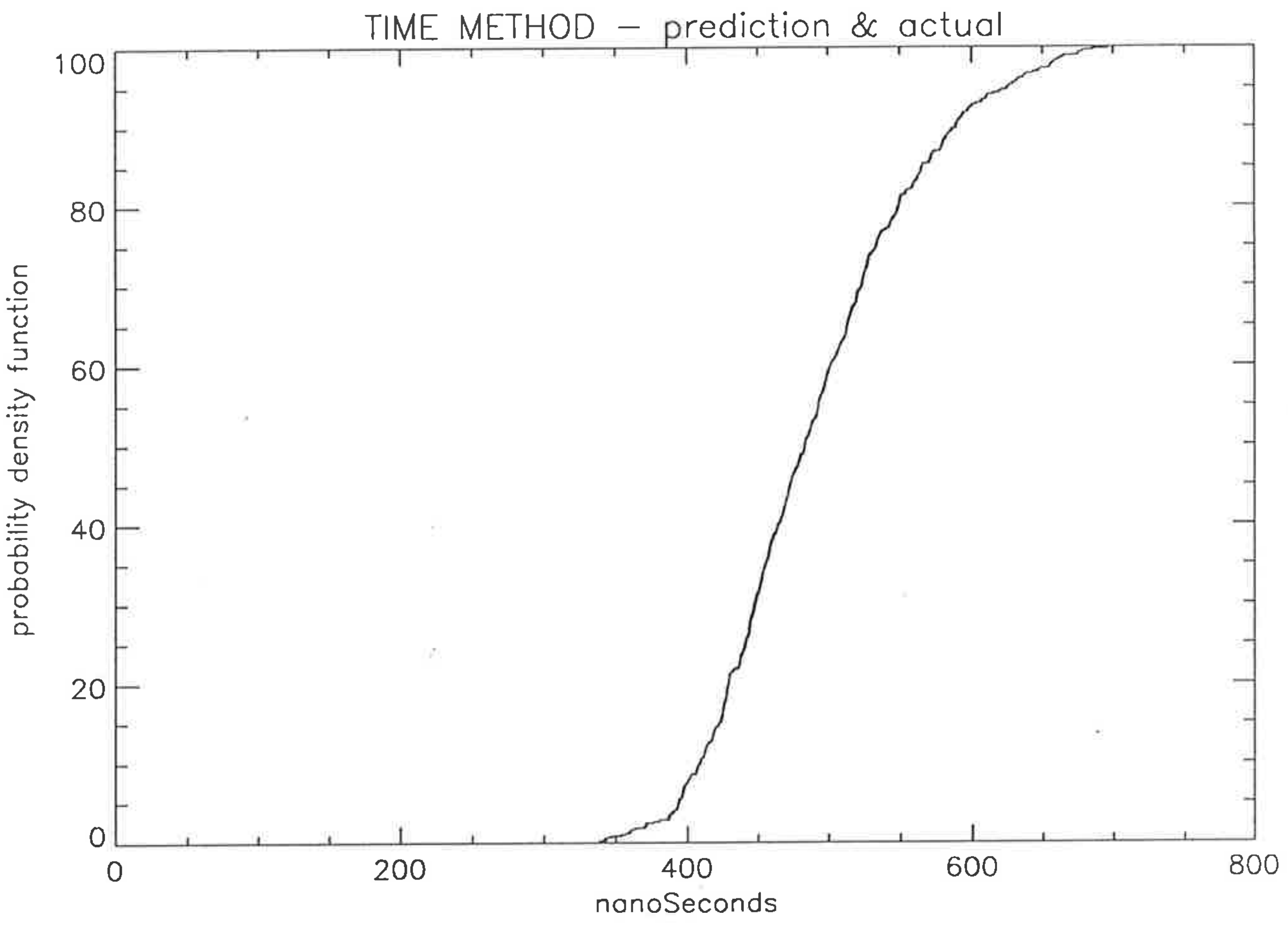


Figure 5.3: COMPARISON OF PREDICTED AND ACTUAL DISTRIBUTION

Further experiments were carried out which examined the effect of sample size (i.e., the number of Monte-Carlo SPICE simulations) on how well the predicted distribution fitted the actual distribution. The TIME method quickly reached a high level of agreement and maintained it as the sample size increased. For a sample size chosen at random between 25 and 150 the TIME method has a very high probability of producing a predicted cumulative distribution in good agreement with the actual cumulative distribution for the 8-stage cascaded system.

The TIME method, which requires stochastic simulation of only the first two cells of an n-cell concatenated structure, one at a time, produces the required statistics with great accuracy, and this is done at vastly reduced cost when compared with a full simulation involving the whole n cells. However, the validity of the algorithm depends on the structure not extending too far across the chip, in order that parameters remain essentially constant across the structure. This is usually the case. If it is not, then the methodology used in the TIME method simulations can also be used to carry out this simulation, i.e., parameters varying steadily across the chip, as well.

Chapter 6

Direct Statistical Method

Recognising that exact fabrication knowledge is unobtainable, I decide to look for a new mathematical model which incorporates parameter variations from the outset. I hope to produce the statistics with just one simulation of an enlarged system of equations, but the method turns out to be too inaccurate.

6.1 The Quest for a New Model

In the case of remote fabrication, exact parameter values are unobtainable - only statistical knowledge is really available. This suggests that a new model based frankly on a statistical approach may be more fruitful and could produce a much more realistic simulation than current deterministic ones.

6.1.1 Deterministic Models

The equations of development of voltage and current in circuits are effectively given as a set of first order, ordinary differential equations in the state variables, voltage and current.

They are:

$$\frac{dx}{dt} = f(x,t)$$

where:

x is a column vector containing the state variables, and

$f(x,t)$ represents a column vector containing functions of the above variables and the time.

There are two major aspects which have not been addressed in these equations.

First, each of the state variables is not known exactly at the beginning of the simulation. This uncertainty in the initial conditions leads to an uncertainty in the final values of the state variables.

Second, many of the fabrication parameters (not state variables) occurring in the equations are not only imperfectly known, but if the same specified simulation was to be run many times, then each time their complete time-history would be different as different actual values are used in the simulation. Each parameter is really only known statistically, i.e., the mean value is specified and the variance about the mean is known or at least estimable.

The effect of this is also to introduce a statistical uncertainty into the state variables.

If each state variable is initially a random variable, then the whole simulation becomes a stochastic process, whose value at any time is also a random variable - that is, each component of the state is defined no longer by a definite number, but by probability distribution. This gives the probability of that variable having a value which lies in a small interval around a definite number.

By treating circuit analogue behaviour as a stochastic process rather than a deterministic one, it should be possible to take account of the influence of variations in fabrication properties and initial conditions on the timing behaviour of the circuit. It appears that in order to gain reliable simulation of the effects of these uncertainties, stochastic differential equations have to replace deterministic ones.

6.2 Equations of the New Mathematical Model

The new equations are thus to be sought in the literature of stochastic processes and a suitable formulation for the present purpose has been presented by Sage and Melsa [SM71].

The first stage in the derivation is to obtain the equation for the time-development of the probability density function for the state vector defined by the set of equations which determine the voltage and current at each node in the circuit.

Then, assuming that each state variable starts out as a gaussian distribution, characterised by a mean value and a variability about that mean value, and in addition assuming that it continues to develop approximately as a gaussian distribution, a useful approximation can be derived which defines the mean value and the variance for each variable at all times. The distribution of finishing times can then be obtained from this by integration.

The interesting points turn out to be:

- that the equations for the *mean* state are the same as already occur in the existing models for the *deterministic* state, and so all previous predictions may in fact be re-interpreted as predictions of the average value instead;

- and that the equations for the covariances are decoupled from the mean state equations, and so may be appended to the existing models without disruption.

In addition, the uncertainties in all fabrication parameters may be conveniently modelled by taking them into the definition of the state (although this creates some extra variables) with equations of the form $dc/dt = 0$, since, although they are initially sampled from gaussian distributions, they do not thereafter alter during the simulation. Even though they do not themselves alter, the effect of the variability of theses parameters is transferred to the variance of the other variables *via* a matrix occurring in the set of equations, and through the off-diagonal elements of the variance matrix.

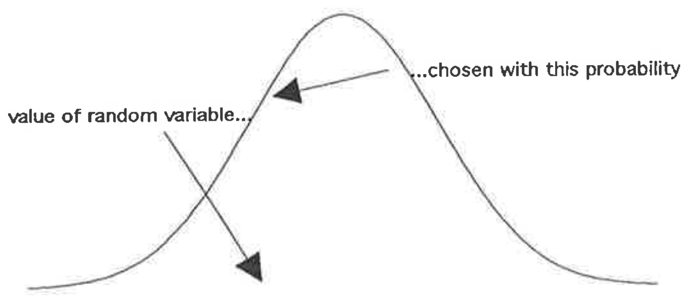
The main variations ocurring in fact and also considered in the simulations have already been summarised in the table on page 28.

6.3 Representation of Uncertainties

It may be seen from the previous discussion that there are two sets of uncertain values: the initial values of the state variables, and the value of parameters in the equations for the state variables. A study of the theory of probability (such as is covered by Papoulis [Pap91]) makes clear that: the initial values may be represented as random variables; the parameters and the state variables may be represented as stochastic processes.

6.3.1 Random Variables

A random variable is a real function whose value is determined by the outcome of a random experiment.



In our case, the initial values of state variables may equally probably be more or less than a mean value. The values of the variables are continuous, and so the most useful representation is the probability density function, which specifies the probability that the random variable lies between two very close values.

More formally, if the random variable may assume a value x and $p(x)$ is its probability density function then $p(x)dx$ is the probability that the value lies between x and $x + dx$. For gaussian distributions, important later,

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-(x-\hat{x})^2/2\sigma^2}$$

where:

$\hat{x}$ = mean value of variable x;

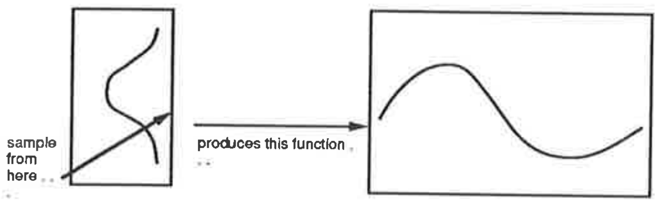
σ^2 = variance of distribution about the mean value.

6.3.2 Stochastic Processes

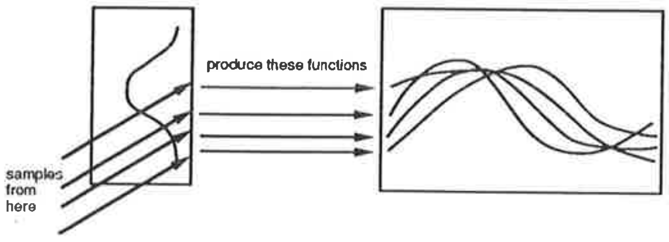
A stochastic process is an infinite collection of random variables

$$\{x(t), t\epsilon[0,T]\}$$

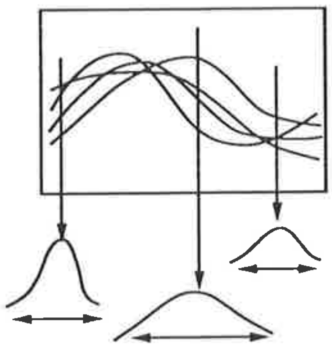
describing the evolution of a natural phenomenon. That is, a stochastic process is a time-function, depending on the outcome of an experiment.



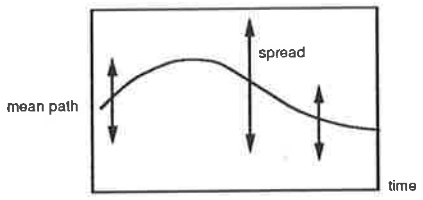
If the experiment is carried out many times, then many different time-functions result, as illustrated.



At any given time, the value of the process is a random variable; it is the probability- density-function composed of all possible time-functions passing through that particular value of time, seen below.



Ultimately the stochastic process is described by the change of the mean value and the standard deviation of a random variable with time, as below.



6.4 Treatment of Uncertainties

The original deterministic equations of motion have been analysed, and two groups of variables identified:

- the state variables, which are stochastic processes, starting out from their initial conditions, which are random variables;
- fabrication parameters in the equations, which are random variables but which are constant throughout the simulation.

6.5 Continuous Stochastic Dynamical System Model

Current deterministic models take no account of the effect of uncertainties in initial states or parameters. This new model does so directly. It achieves this by treating leaf-cell transistor simulation as a stochastic process. Equations for propagation of variance produce the necessary performance statistics during each simulated run.

The model aims to produce, in one trajectory, what the Monte-Carlo approach produces in many. It takes its equations for the state of the circuit from the statistical literature [SM71]. The equations for the time-development of the state are of the same form as in the deterministic model.

It is assumed that each state variable commences as a gaussian distribution, characterised by a mean value and a variability about that mean value, and continues to develop approximately as a gaussian distribution. These equations, along with variance equations appended to the state equations, produced the mean state and the variance directly as functions of time for each variable.

6.5.1 Stochastic Differential Equations

The equations governing this set of stochastic processes may immediately be written down in a general form.

If there are n state variables, and m parameters which have been singled out for random treatment, then for t≥0,

$$\frac{dx(t)}{dt} = f(x(t), t)$$

where:

x is an n by 1 column vector containing the state variables;
f is an n by 1 column vector of function of the x variables;
t is the time.

Following the methods and terminology used by Sage & Melsa [SM71] and Jazwinski [Jaz70], this is called a continuous stochastic dynamical system of equations, or simple *a CSDS Model.* The initial conditions for the equations are the values of the state variables each expressed as a random variable with a given probability density distribution.

6.6 Solution of Equations

In this section the general solution of the stochastic dynamic system of equations is derived. It turns out that in order to obtain practical solutions various approximations have to be made. Instead of obtaining the full probability density function as a function if time, only the mean and variance as functions of time can reasonably be calculated, and for gaussian distributions this is all that is needed to give a reliable picture of the simulation process.

6.6.1 The Probability Density Function

If the deterministic simulation is run over and over, using random samples of the stochastic processes representing the parameters, then the statistics of the state vector may be compiled in terms of its probability density function of time.

The same function may be obtained theoretically under certain general assumptions, the chief of which is that the process is a Markov process, which the present one clearly is, as described by Arnold [Arn74].

Gilman and Skorohod [GS79] discuss the mathematical conditions for uniqueness and existence of this solution, and all the functions used in the present case satisfy them.

Jazwinski [Jaz70] gives a readable derivation and McGarty [McG80] gives some useful examples as well. However, Sage & Melsa [SM71] follow their derivation with equations for the mean and variance which are more useful for a computer implementation, and their notation is adopted.

6.6.2 Equation for Probability Density Function

The stochastic differential equations have already been derived from the deterministic equations and are $dx(t) = f(x(t), t)dt$.

Using the results for integrating stochastic processes of this type, Ash [AG75], Wong [Won71], Arnold [Arn74], and Sage & Melsa all obtain an equation for the conditional probability density function (given the initial distributions of the state variables) as the function p in the equation

$$\frac{dp}{dt} + trace\frac{\partial(fp)}{\partial x} = 0$$

where $p = p(x,t)$ = joint conditional probability density function such that $p(x,t)dx_1dx_2...dx_n$ is the probability that variable x_1 lies between x_1 and $x_1 + dx_1$, etc., given the initial distributions of $x_1, ..., x_n$;
$f = f(x,t)$ has already been defined;
'trace' means the sum of the diagonal elements of a matrix.

In the system to which this equation will be applied, *i.e.,* the multiplier leaf cell used as an example, the vector x contains about 100 nodes and thus about 200 state variables, *viz.,* the voltage and current at each node. Even if this partial differential equation is solved, the result is a function of 200 variables at any one time. Just to specify this function is quite impractical. Fortunately, it is only necessary to know the development of the mean values of the state variables, along with an estimate of the variability to be expected about this mean value, in order to gain a clear picture of the trajectory. Sage and Melsa[SM71] go on to give a useful approximation for these two quantities.

However, it should be noted that on some non-linear stochastic systems of the type studied here, initially-normal distributions may, under some circumstances, develop into markedly non-normal, multi-modal distributions as time progresses. The mean and variances then do not of course give, by themselves, a good representation of the process, and care must be exercised in the interpretation of the results.

6.6.3 Equation for the Mean

Once the equation above is derived, the mean value of x as a function of time may be approximately obtained by: transforming the stochastic differential equation, multiplying by $x(t)$, and averaging by integration. To complete the integration the vector function $f(x,t)$ is expanded about the mean value of x (called $\hat{x}$) as a Taylor series, on condition that no particular $x(t)$ ever gets very far from the mean. The resulting integration introduces the variance matrix from cross product terms such as $x_i x_j$. The result is approximately

$$\frac{d\hat{x}}{dt} = f(\hat{x}(t),t) + \frac{1}{2}V(t) : \partial f(\hat{x}(t),t)$$

where:

$\hat{x}(t)$ = a column vector containing the mean values of each corresponding state variable;

$f = f(x,t)$ are the original (deterministic) functions;

$f(\hat{x},t) is f(x,t)$ evaluated at $x = \hat{x}$;

$V = V(t)$ is the covariance matrix (ik element=$E(x_i - \hat{x}_i)(x_k - \hat{x}_k)$))

$F = \partial f/\partial x$ (ik element=$\partial f_i/\partial x_k$);

$V : \partial = \sum_{ik} V_{ik}\partial^2/\partial x_i \partial x_k$ is an abbreviation for this operator.

6.6.4 Equation for Variance

A useful approximation for the variance matrix $V(t)$ is obtained by Sage and Melsa ([SM71], who expand the expression for the variance

$$[x(t) - \hat{x}(t)][x(t) - \hat{x}(t)]^T$$

then multiply by the probability density function defined by the equation, integrate over all the x variables, and expand $f(x,t)$ in a Taylor series about the mean $x = \hat{x}$, producing the propagation equation for the variance

$$\frac{dV(t)}{dt} = \frac{\partial f(\hat{x},t)}{\partial \hat{x}}V(t) + V(t) : \frac{\partial f^T(\hat{x},t)}{\partial \hat{x}}$$

where the ':' notation has already been defined.

6.7 Implementation of the Solution

The notation for mean value will be dropped, since all state variables are now mean values.

6.7.1 Propagation Equations

The equations are:

$$dx/dt = (1 + \frac{1}{2}V : \partial)f \tag{6.1}$$

$$dV/dt = FV + (FV)^T \tag{6.2}$$

Each diagonal element of the matrix V contains the variance of the corresponding state variable, interpreted as a near-gaussian distribution about the mean value, x.

6.7.2 Treatment of 'Constant' Random Variables

All of the fabrication parameters in the equations (such as the capacitance) do not change during the trajectory but are sampled from an initial distribution at $t = 0$ (i.e., at fabrication). The correct method of treatment for these is to declare them as state variables with no rate of change. In this way the influence of their variability can be transferred to the variance of the other state variables *via* the F matrix and the off-diagonal elements of the variance matrix. This is a general feature of the equations.

The quantities in question have already been listed on page 28 and this yields an extra 15 state variables with equations of the form $dx/dt = 0$.

6.7.3 Number of equations to integrate

For a system with n state variables there are n equations for the state variables, plus the equations for the terms of the variance matrix, $V(t)$, to be integrated. This is a symmetric matrix, so the whole lower diagonal part may be ignored, giving just $(n * n - n)/2 + n$ equations for the variance.

Thus the total number of equations to be integrated is

$$n(n+1)/2.$$

In the case considered here, n is about 200, so there are 20,000 equations. This is a massive system to implement. Before doing so, there are some tests that can easily be carried out to determine whether it would be worth-while.

6.8 Direct Statistical Model Results

The voltages at nodes inside a leaf cell and the variances of those voltages may be obtained directly in one integration of this system of simultaneous equations describing the analogue behaviour of the digital circuit (the leaf cell), viz., equation 6.1 and equation 6.2 on page 78.

From these equations, it can be appreciated that for this method to work, three tests must be satisfied:

1 the first equation is a good approximation, *i.e.,* no $x(t)$ gets too far from the mean of all the simulations. This means that the deterministic simulation starting with the mean values must be identical to the mean of all the simulations.

2 the assumptions used to derive the approximate, practical solution are valid, *i.e.,* the distributions start and continue as near-gaussians.

3 using the real mean and variance (from the experiments) in the equations must produce the actual cumulative distribution function.

To assess whether this would work in practice, without actually implementing the extra equations, imagine that they have in fact been implemented and integrated to obtain the best possible values[1].

The direct method of obtaining the statistics is as follows. If the outputs of the cell mentioned above are called C,S,X,Y and the voltages are c,s,x,y at time t, then the joint probability density function is

[1]i.e. *the actual ones* for x and V , obtained from the 350 runs

$$f_{CSXY}(c,s,x,y;t)$$

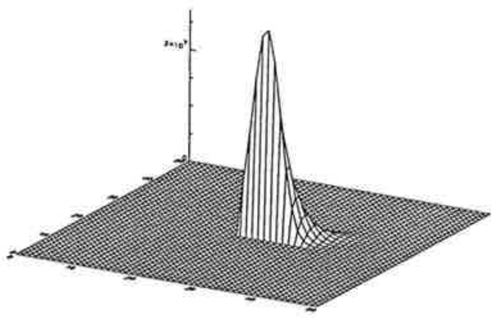
and the required probability

P{carry ready & sum ready & X ready & Y ready at time t}

is given formally [Pap91] by

$$\int_{2.4}^{\infty} dc \int_{2.4}^{\infty} ds \int_{2.4}^{\infty} dx \int_{2.4}^{\infty} dy \, f_{CSXY}(c,s,x,y;t) = F(t)$$

This integration finds the part of the function, illustrated below, which is above the plane and beyond the 'ready' state (2.4 volts).



Now it is not known whether the process really is jointly normal, or if the outputs in fact develop as normal distributions, but absolutely no use of the results may be made unless this is so. If it is so, then it follows that

$$f_{CSXY}(c,s,x,y;t) = exp\{-1/2X^T V^{-1}(t)X\}/\sqrt{(2\pi)^3 \det V(t)}$$

where X is the vector of the carry and sum and V(t) is the covariance matrix just integrated from the equations $dV/dt = FV + (FV)^T$. The function $f_{CSXY}(t)$ may be obtained numerically from this expression.

The best estimates of the mean and variance are the actual mean and variance obtained from the Monte-Carlo simulations already carried out. So a direct comparison may be made of the actual $F(t)$ (straight from the Monte-Carlo results) and the predicted $F(t)$ from the formula (using the best possible value of $V(t)$ constructed from the data). This has been done and the comparison is shown in figure 6.1 on page 81.

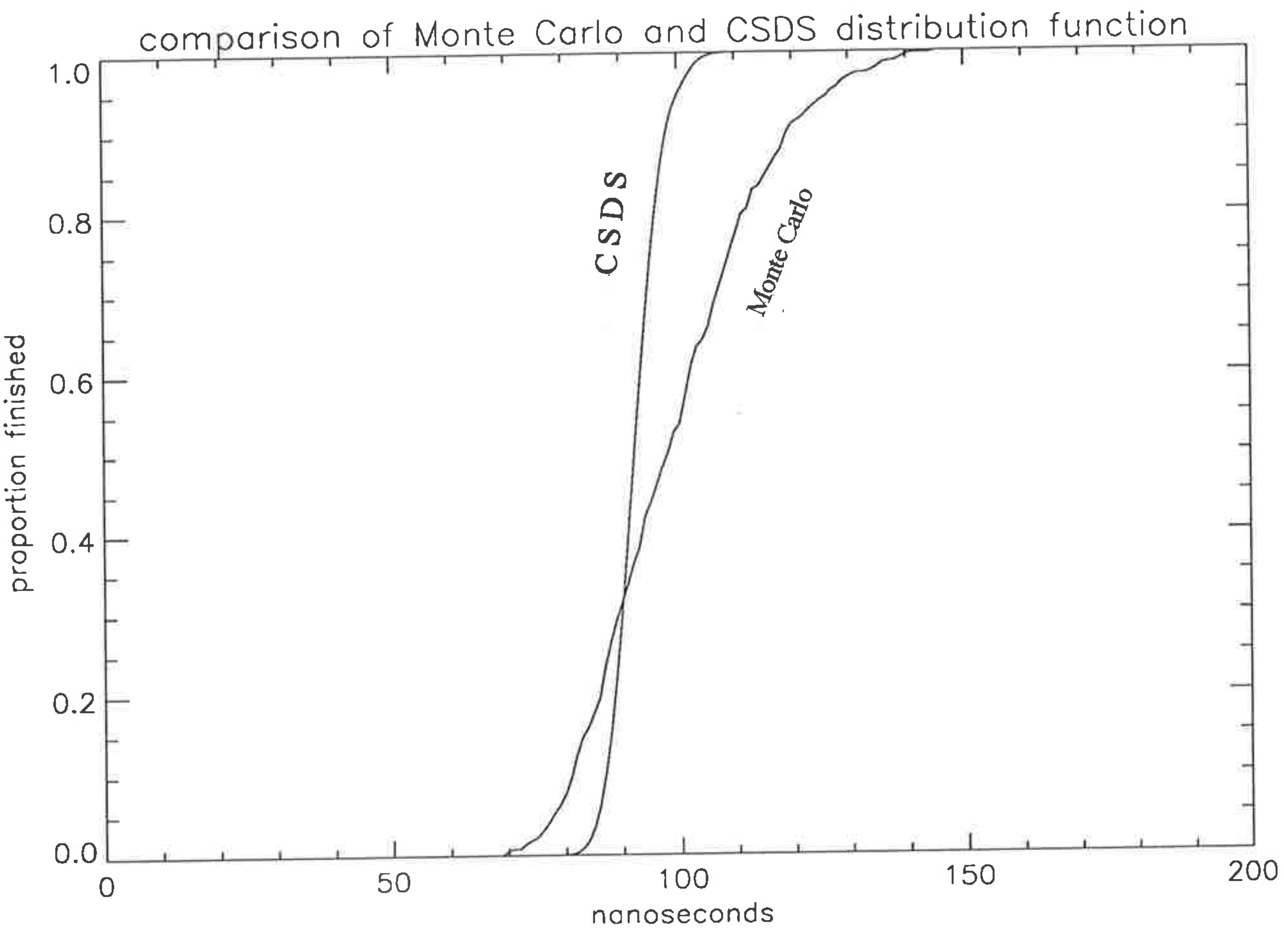


Figure 6.1: Comparison of CSDS and MonteCarlo Distribution Functions

6.8.1 Linear vs Non-linear?

The next question of interest is: do the linear equations apply, or do the non-linear corrections need to be added? This is easily answered, because the equations for the mean trajectory in the CSDS case are the same as those in the deterministic case, provided they start with the mean values and there are no non-linear corrections. If they are not the same, then non-linear corrections are needed.

It appears that this term can be ignored, except in the cases where the variances are getting large, (especially where they are initially large); it may then make an important difference to the dx/dt equation, producing a mean trajectory that is not the same as a deterministic one starting with the mean initial values.

The results of this comparison are shown: for the sum, in figure 6.2 on page 83; for the carry, in figure 6.3 on page 84.

6.8.2 Agreement between Methods

The second question of interest is: does the CSDS model give practically the same results for the mean trajectory and its dispersion as a simulation by Monte-Carlo methods? This is answered by solving the system by both methods once again. The sum and carry are plotted as their equivalent gaussian probability- densities, with the Monte-Carlo (350 runs) above and the CSDS (1 run) below: for the sum, in figure 6.4 on page 85; for the carry, in figure 6.5 on page 86.

6.9 Comparisons and Conclusions

The results of the two approaches are presented in graphs of the probability-density-functions for the sum and carry voltage. The Monte-Carlo results are compiled from 350 trajectories with initially-sampled gaussian states; the continuous stochastic dynamical system (CSDS) model results come from a

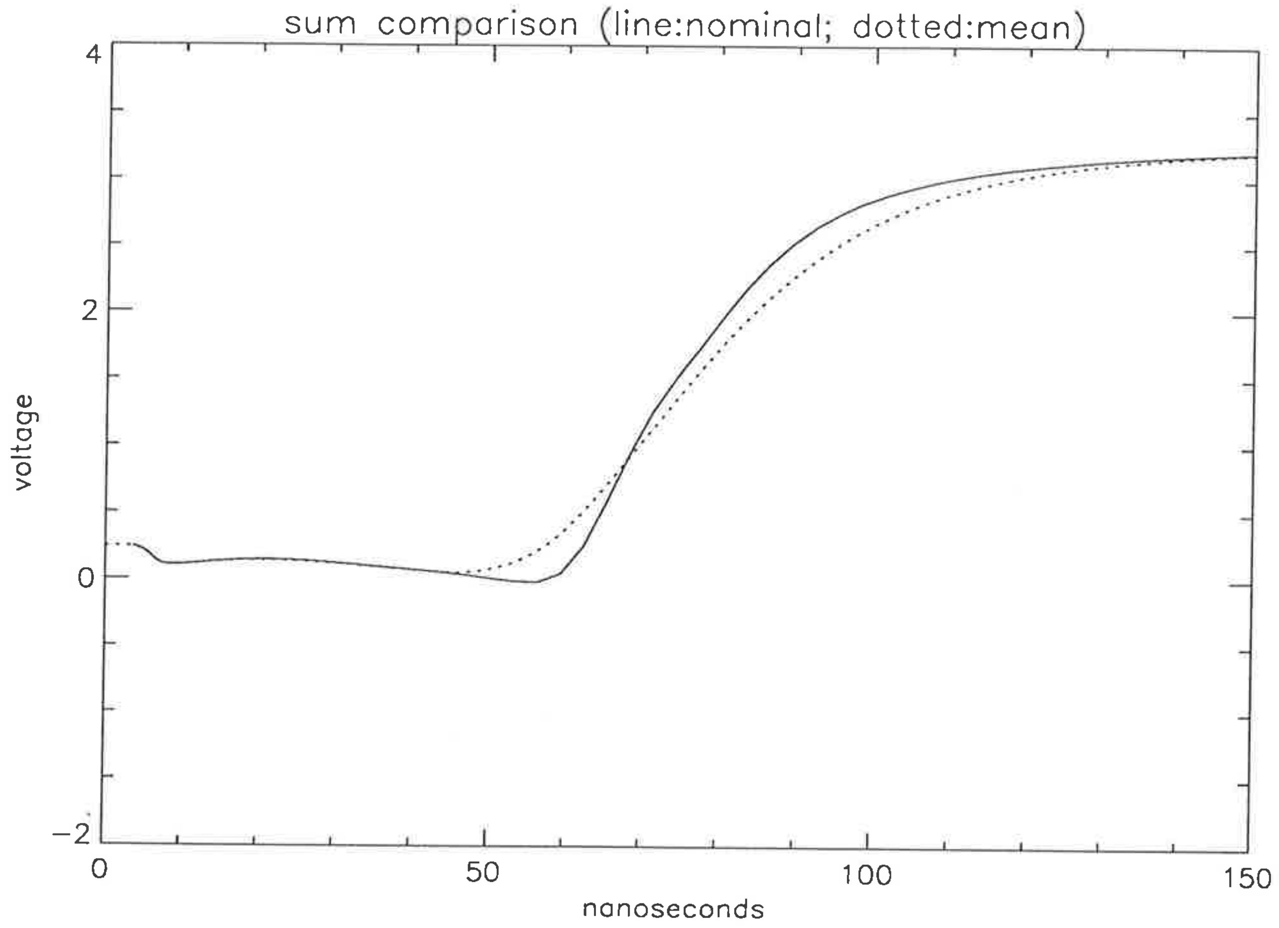


Figure 6.2: Sum: Mean (dotted) and Nominal Trajectories

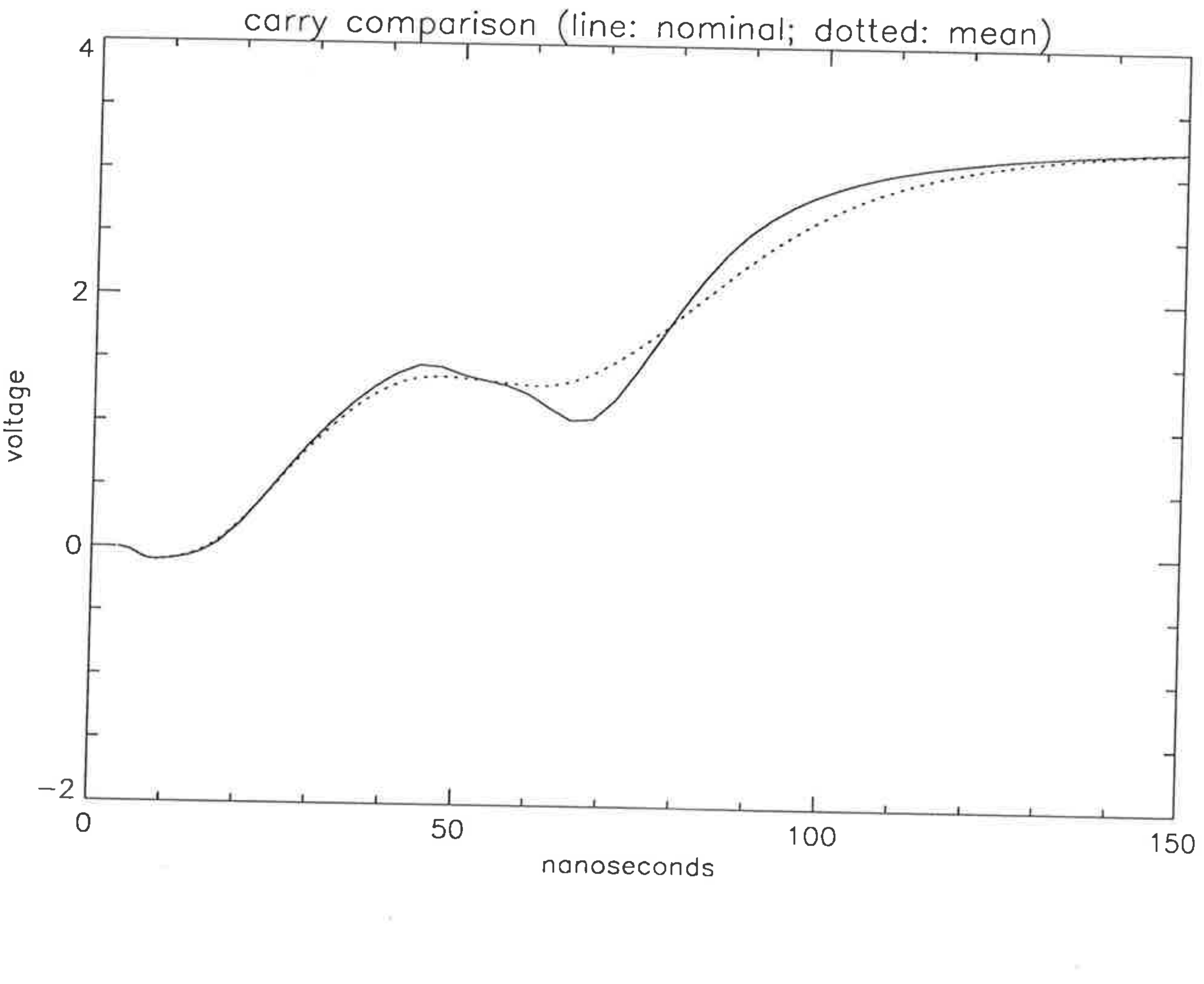
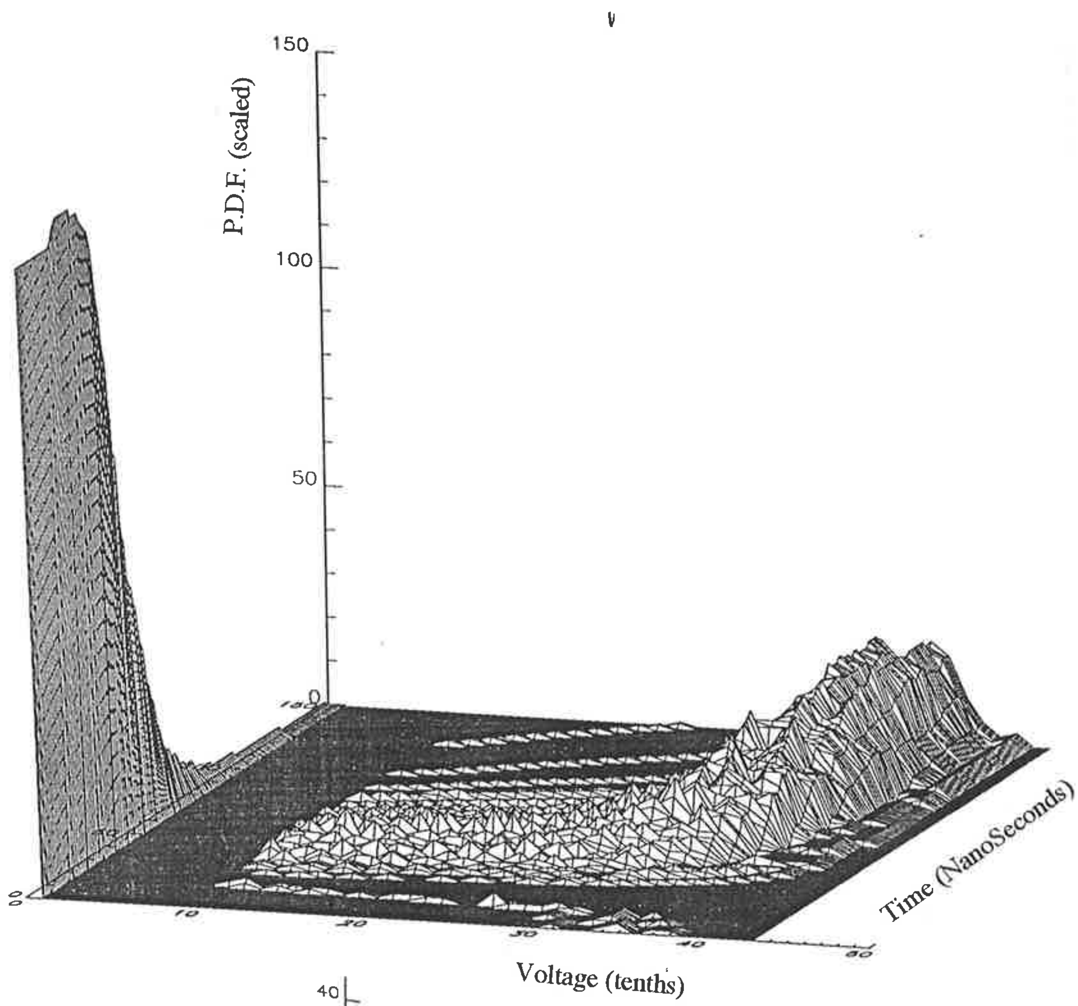


Figure 6.3: Carry: Mean (dotted) and Nominal Trajectories



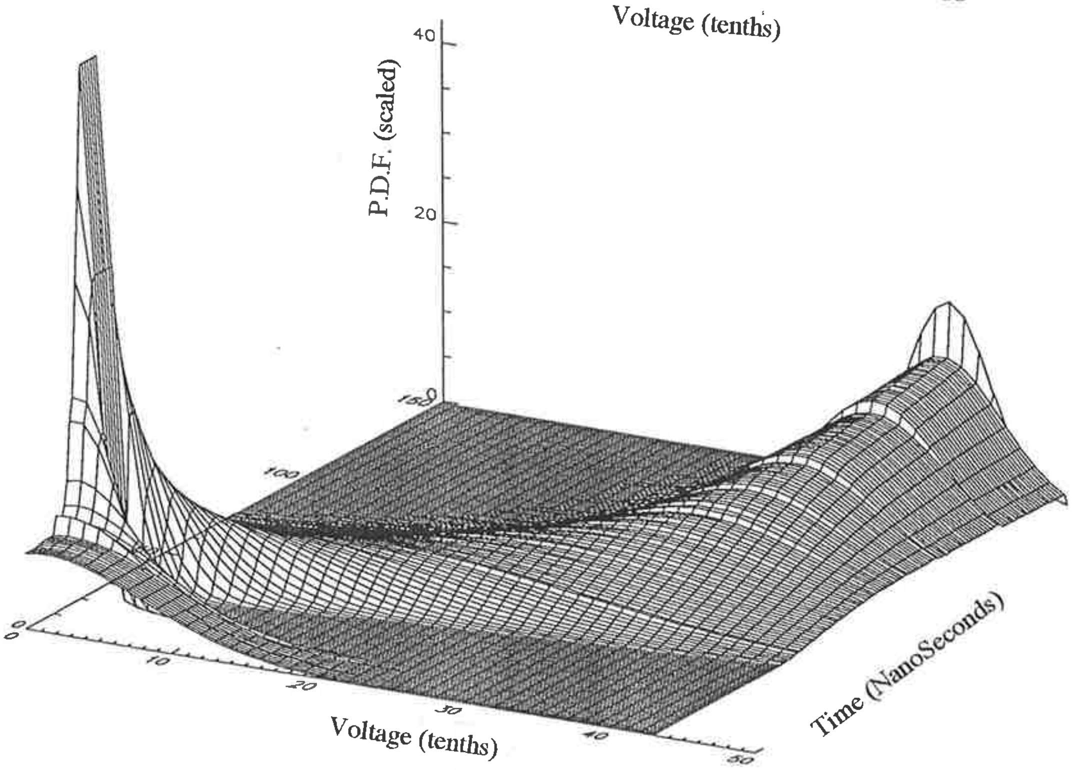
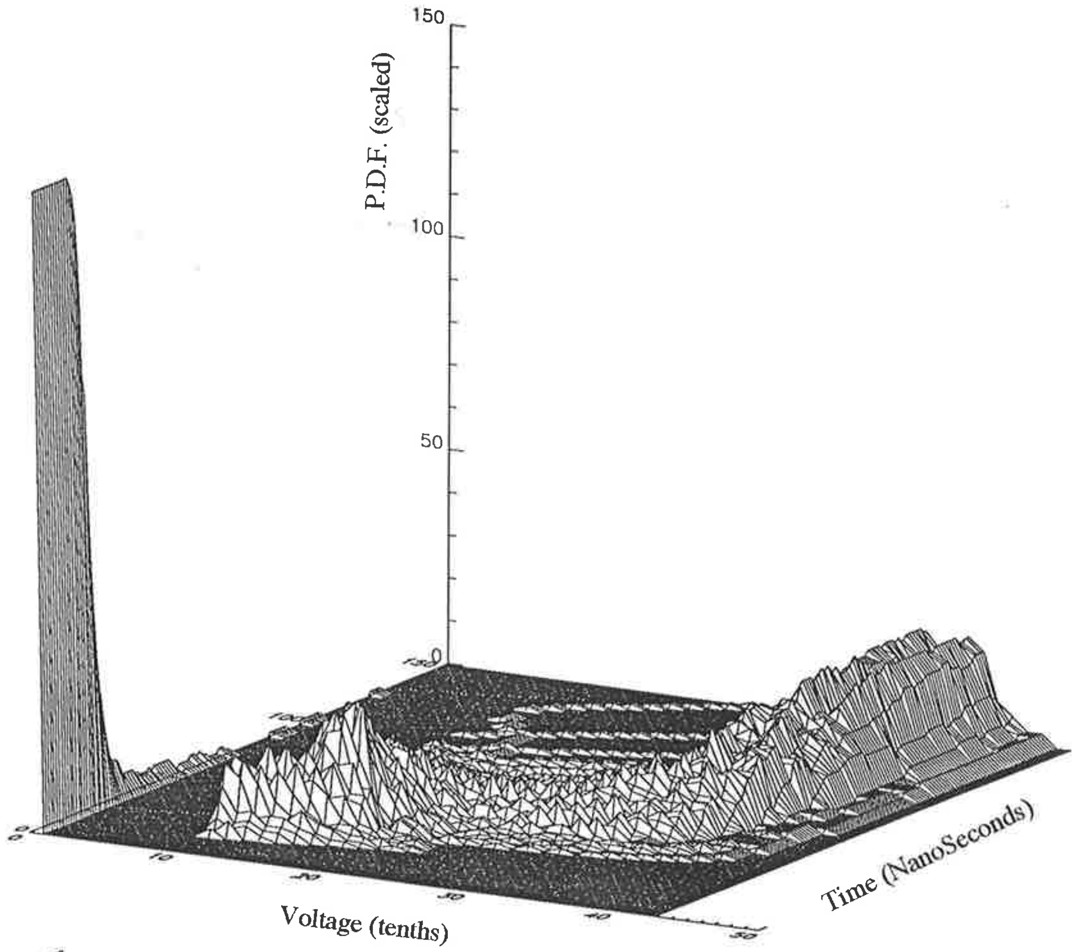


Figure 6.4: SUM: MONTECARLO ABOVE, CSDS EQUIVALENT BELOW



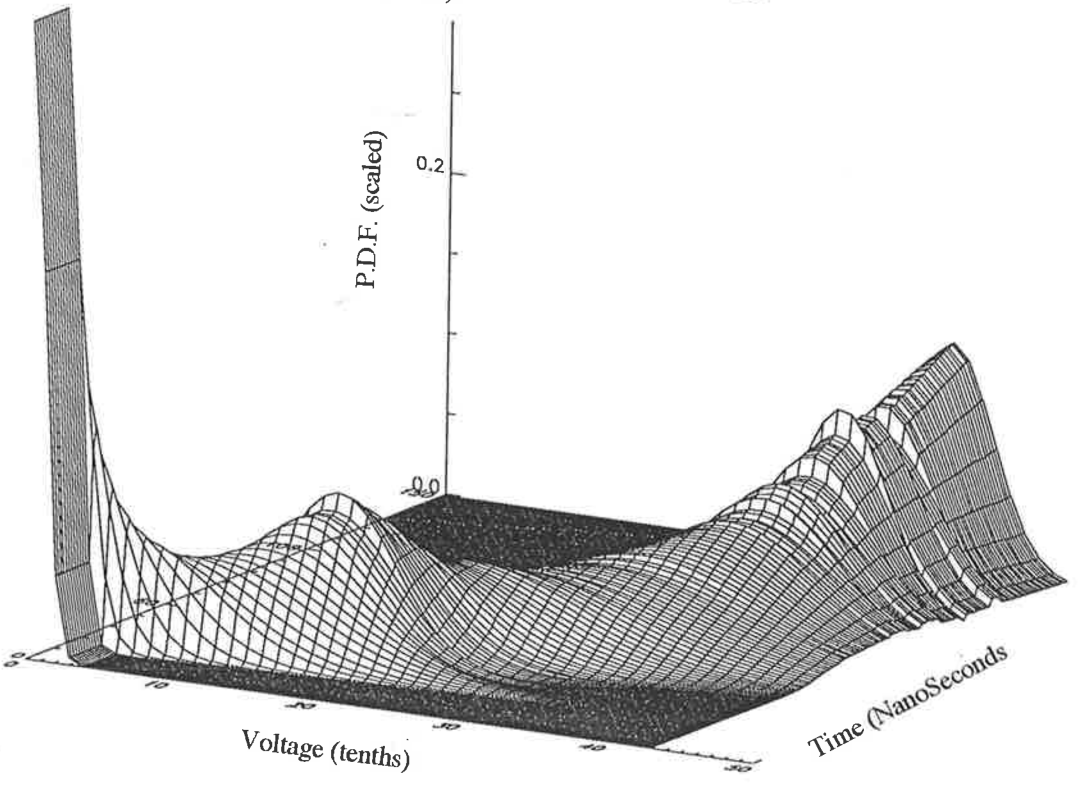


Figure 6.5: Carry: MonteCarlo above, CSDS Equivalent below

single trajectory using the same initial Gaussian distributions. Reference to these graphs makes it clear that the mean values are well represented by the CSDS model, but that the scatter about the mean value (the dispersion), whilst remaining gaussian and of approximately the correct form, is too large at each time step.

Chapter 7

Parameter Fitting Method

Another likely approach is to assume that the spread of finishing times is, for one reason or another, well represented by a known function. Here I discover that this function is closest to the Erlang distribution, and that the number of necessary simulations might be reduced somewhat by knowing this.

7.1 Fitting to Known Functions

The form of the distribution of finishing times has already been obtained for the single-cell case and the concatenated-structure case. To match the form of a known function to these cases it is necessary to match measures of the forms. The best measures of the form of a distribution function are the central statistical moments: mean, variance, skewness and kurtosis. The parameters of the distribution can be calculated from the estimated measures. It might be possible to get a good estimate of these measures by running just a handful of simulations. This would provide a significant time saving over the methods already described.

7.1.1 The Central Statistical Moments

Where finishing times from N simulations are gathered, the most important measures of the form of the distribution function are:

the average value of the time

$$mean = \mu = \frac{1}{N}\sum_{i=1}^{N} x_i$$

the concentration of times near the average

$$variance = \sigma^2 = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^2$$

how unsymmetrical the distribution is

$$skewness = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^3$$

and the character of any tails of the distribution

$$kurtosis = \frac{1}{N}\sum_{i=1}^{N}(x_i - \mu)^4.$$

The cumulative distribution function $F(t)$ is the probability that the time is at least equal to t and it is formed by integrating these probability density functions

$$F(t) = \int_0^t f(\tau)\, d\tau.$$

where

$f(t)$ = the probability that the values lie in an interval δt about t.

Taking N samples, with replacement, from an effectively infinite population gives the average value of the mean is μ and the variance of the mean is σ^2/N. It has been seen that recommendations average $N = 500$.

7.1.2 Restriction on Distribution Functions

From the above definitions it can be seen that an unusually large sample value would distort the skewness and kurtosis much more than it would distort the mean and variance. This happens because of the higher powers involved. This means that attempting to fit a proposed function based on these higher moments would lead to a very misleading result.

Thus, if a small number of samples is desired, it is much safer to only consider distributions which are completely determined by the two low order independent parameters, the mean and variance.

Suitable distribution functions of this type may be found in the comprehensive *Table of Probability Functions* [Luk75] and include: *erlang, beta family, F(k,m) family, lognormal, gaussian, pearson type 3, weibull* and *extreme value.*

Of these, the only ones found to come close to being good fits were: the *Beta, Normal, Extreme-Value* and *Erlang* distributions. These are studied in detail below.

7.2 Possible Distribution Functions

7.2.1 the Beta distribution

parameters α, β

$$mean = \frac{\alpha}{\alpha + \beta}$$

$$variance = \frac{\alpha\beta}{(\alpha+\beta)^2(\alpha+\beta+1)}$$

$$f_{\alpha\beta}(t) = \frac{t^{\alpha-1}(1-t)^{\beta-1}}{B(\alpha,\beta)}$$

with the Beta function

$$B(\alpha,\beta) = \frac{\Gamma(\alpha)\,\Gamma(\beta)}{\Gamma(\alpha+\beta)}.$$

The time axis is scaled to the interval [0,1] and the mean and the variance are scaled accordingly.

7.2.2 the Normal (Gaussian) distribution

parameters μ, σ

$$mean = \mu$$

$$variance = \sigma^2$$

$$f_{\mu\sigma}(t) = \frac{1}{\sigma\sqrt{2\pi}} exp[-\frac{1}{2}(\frac{t-\mu}{\sigma})^2]$$

7.2.3 the Extreme-Value distribution

parameters α, β

$$mean = \alpha + \gamma\beta$$
$$variance = \frac{(\pi\beta)^2}{6}$$
$$\gamma = Euler's\,constant = 0.5772156649...$$
$$x = \frac{t-\alpha}{\beta}$$
$$f_{\alpha\beta}(t) = \frac{1}{\beta} e^{(-x-e^{-x})}.$$

7.2.4 the Erlang (Truncated Gamma) distribution

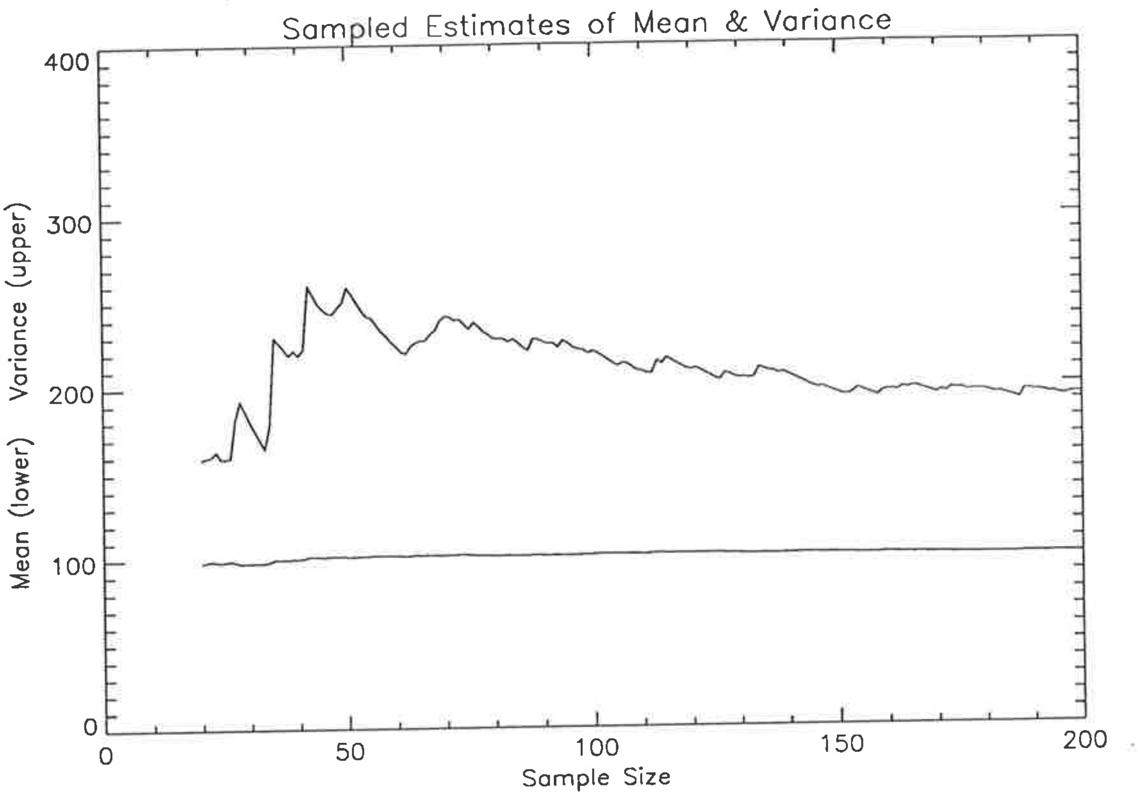
parameters α, k

$$\alpha = \frac{1}{\mu}$$
$$k = (\frac{\mu}{\sigma})^2$$
$$f_{\alpha k}(t) = \begin{cases} (\alpha k)^k\, t^{k-1}\, e^{-\alpha k t} / \Gamma(k) & \text{if } t > t_{critical} \\ 0 & \text{otherwise} \end{cases}$$

7.3 Experimental Data

7.3.1 Single Cell

The mean finishing time for the single leaf cell was $\mu = 100nSec$ and the variance $\sigma^2 = 202nSec^2$. This was based on 500 simulations. The graph at the top of figure 7.1 on page 92 shows the mean (lower line) and the variance (upper) of the finishing times as more and more simulations are done. The estimate of the mean time settles very early, but the estimate of the variance is quite erratic until at least 100 simulations have been done, and really only gets near the population value after 150 simulations.



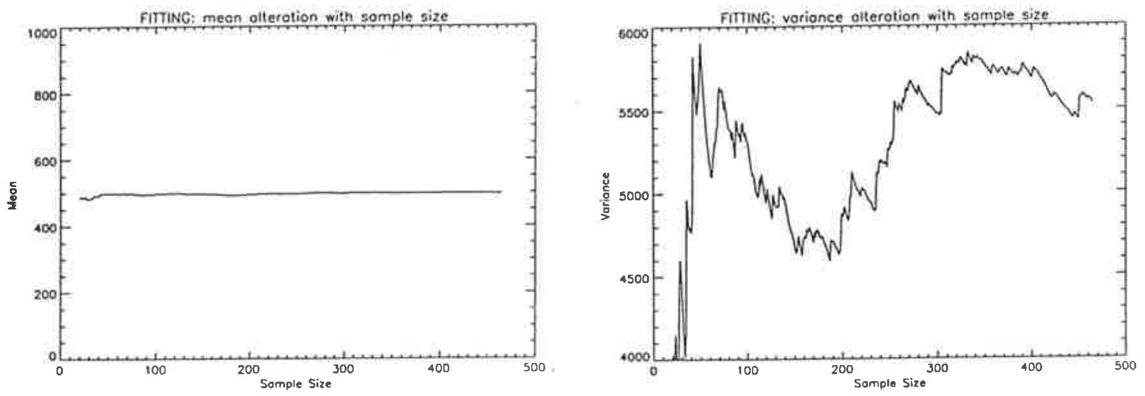


Figure 7.1: MEAN & VARIANCE: SINGLE (TOP) & CONCATENATED (BOTTOM)

7.3.2 Concatenated Structure

The mean for the concatenated 8-cell ripple-carry adder structure was $\mu = 497nSec$ and the variance was $\sigma^2 = 5947nSec^2$. This was based on 465 simulations.

The graph at the bottom of figure 7.1 on page 92 show the mean (left) and the variance (right) of the finishing times as more and more simulations are done. The estimate of the mean time settles early, but the estimate of the variance is erratic and only gets near the population value after 300 simulations.

7.4 Results of Fitting

The experimental results from these simulations were used to estimate the population statistics (mean, variance, skewness and kurtosis). Then the distribution functions were plotted against the data to give an initial indication of goodness-of-fit. These are summarized in the tables below.

7.4.1 Single Cell

	experimental	100.341	202.233	1706	124039
FUNCTION	*parameters*	Mean	Variance	Skewness	Kurtosis
Beta Family	$\alpha = 7.2$	100.3	202.2	-5	117925
	$\beta = 7.82$				
Normal	$\mu = 100.3$	100.3	202.2	0	122695
	$\sigma = 14.22$				
ExtremeValue	$\alpha = 93.4$	100.3	202	3187	209992
	$\beta = 11.08$				
Erlang	$\mu = 0.010$	100.3	202	815	127622
	$k = 49.786$				

The top four graphs in figure 7.2 on page 94 show the proposed fit to the probability density function plotted with the data for the *Beta, Gaussian, Extreme-Value* and *Erlang* distributions. The *Beta* and *Gaussian* are ruled out at this stage, but the *Extreme-Value* and the *Erlang* look promising.

The bottom four graphs in figure 7.2 on page 94 show the proposed fit to the cumulative probability function plotted with the data for the *Beta, Gaus-*

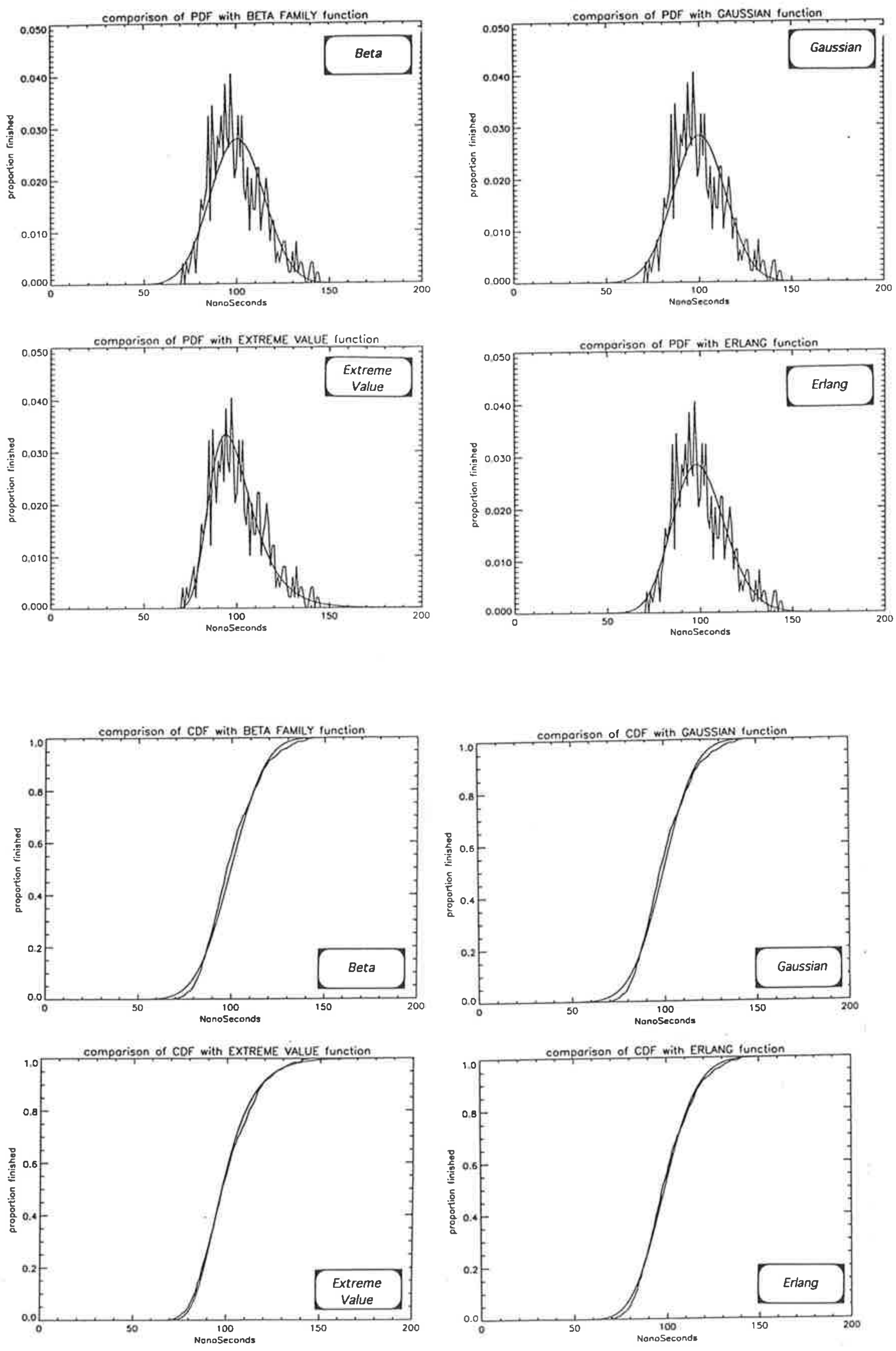
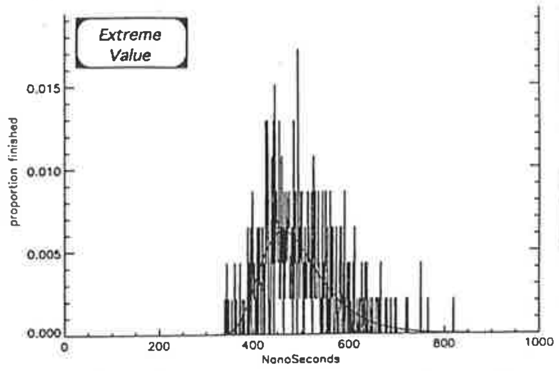


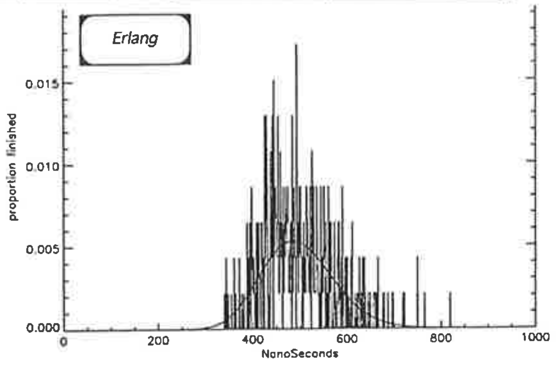
Figure 7.2: Best Fits: P.D.F. (top) and C.D.F. (bottom)

sian, *Extreme-Value* and *Erlang* distributions. The *Extreme-Value* and the *Erlang* both show small discrepancies near the extremes of the distributions.

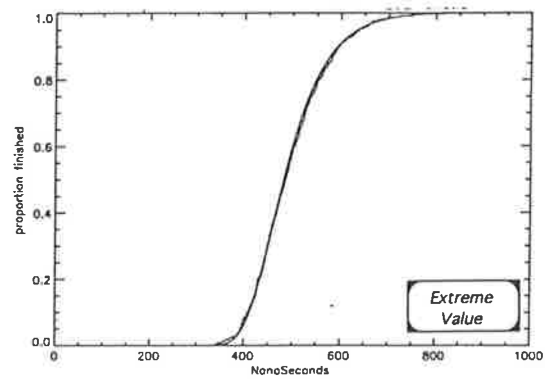
7.4.2 Concatenated Structure

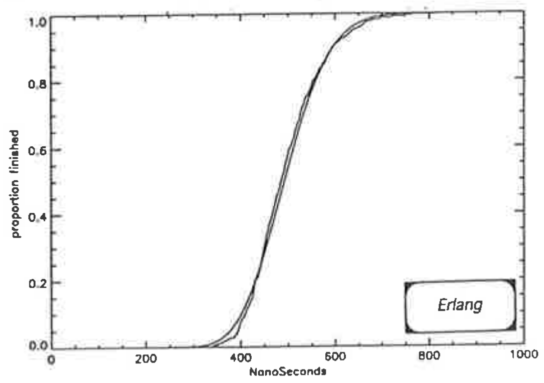
	experimental	496.7	5947	356344	131036672
FUNCTION	*parameters*	Mean	Variance	Skewness	Kurtosis
Beta Family	$\alpha = 16.667$	497.3	5948	-56423	100175564
	$\beta = 11.821$				
Normal	$\mu = 496.7$	496.7	5947	-102	106096589
	$\sigma = 77.1$				
ExtremeValue	$\alpha = 462.591$	496.6	5681	415316	139099393
	$\beta = 60.133$				
Erlang	$\mu = 0.002$	497.3	5939	139416	110001690
	$k = 41.578$				





The two graphs above show the proposed fit to the probability density function plotted with the data for the *Extreme-Value* and *Erlang* distributions. The *Beta* and *Gaussian* are very poor and have been rejected.





The two graphs above show the proposed fit to the cumulative probability function plotted with the data for the *Extreme-Value* and *Erlang* distributions. Both show small discrepancies near the extremes of the distributions.

7.5 Chi-Squared Test of Fitness

These tests by eye have eliminated the obviously-worst fits; however, a χ^2 test is the most appropriate mathematical test of how well the proposed function matches the given distribution [Bra75].

In order to test which, if any, of these functions fits these data the best, the distribution of times is divided into K groups containing the observed frequencies o_i, and the proposed function is used to calculate the expected frequencies e_i. The statistic

$$\sum_{i=1}^{K} \frac{(o_i - e_i)^2}{e_i}$$

is formed and at a predetermined nominal value of probability α with $K-1$ degrees of freedom, the fit is rejected as not a significantly good fit if

$$Probability(\chi^2_{K-1} >= \sum_{i=1}^{K} \frac{(o_i - e_i)^2}{e_i}) <= \alpha.$$

The table following summarizes the results. The *Extreme Value* distribution and the *Erlang* distribution prove to be very good fits to the actual distribution.

function	*cell degrees of freedom*	*value* of χ^2	*concatenated degrees of freedom*	*value* of χ^2
Beta	30	71	79	87900
Normal	35	65	84	311
ExtremeValue	27	29	59	63
Erlang	32	35	74	89

7.5.1 Single Cell χ^2 Result

The top four graphs in figure 7.3 on page 97 show how the value of the critical χ^2 value alters as more simulations are run. It is shown for the *Beta, Gaussian, Extreme-Value* and *Erlang* distributions. The upper line is the χ^2 value itself, whereas the lower line is the critical value below which it must fall if the fit is to be accepted as a good one. The *Beta* and *Gaussian* distributions are definitely unacceptable, but the *Extreme-Value* looks promising in the long run, and the *Erlang* is a significantly good fit after just 30 simulations have been run.

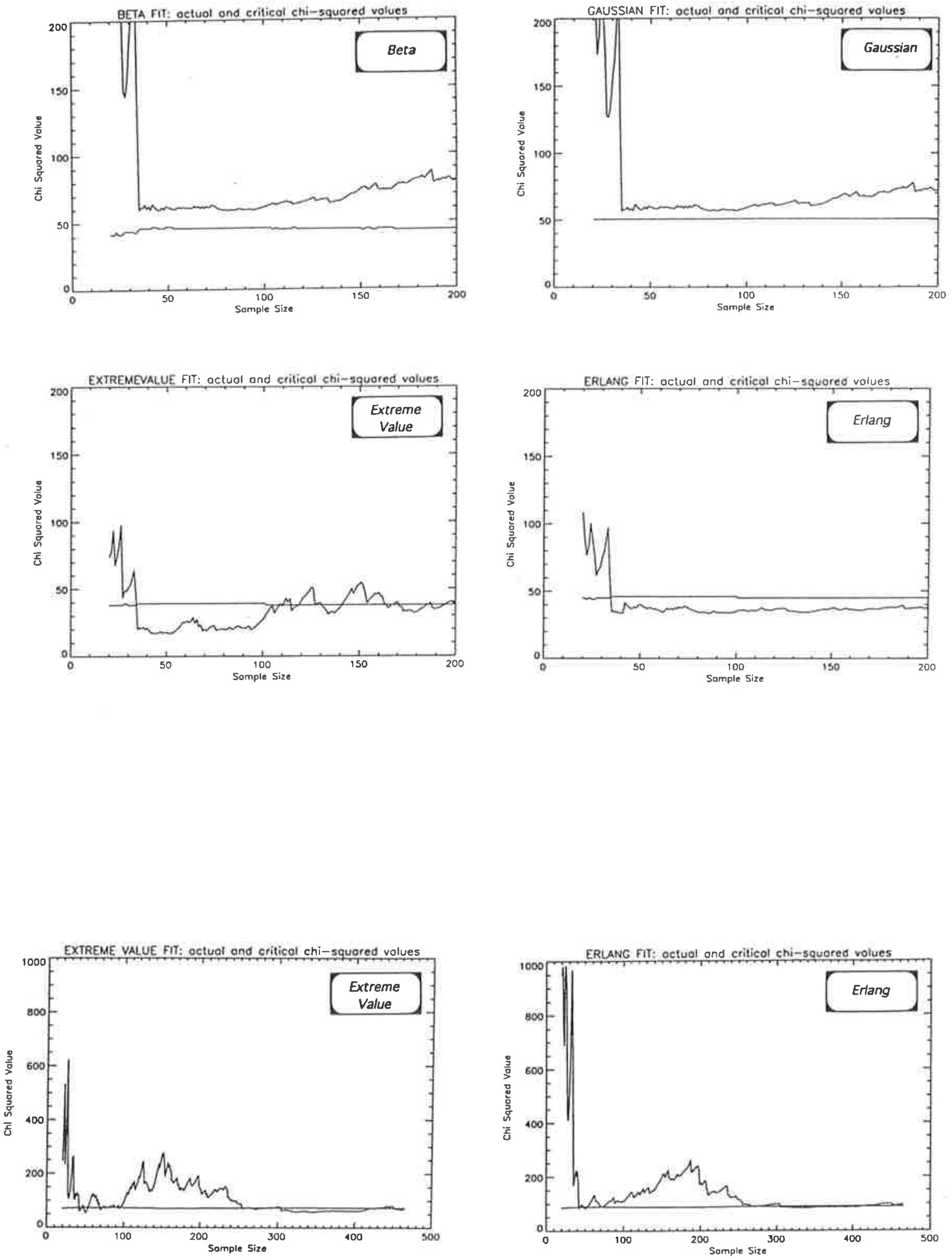


Figure 7.3: χ^2 VARIATION WITH SAMPLE SIZE

7.5.2 Concatenated Structure χ^2 Result

The bottom two graphs in figure 7.3 on page 97 show how the value of the critical χ^2 value alters as more simulations are run, only for the *Extreme-Value* and *Erlang* distributions. Both the *Extreme-Value* and the *Erlang* are accepted as significantly good fits after 250 simulations. and as usefully good fits with only 30 to 100 simulations.

7.6 Conclusions

The results are consistently better for the Erlang distribution than for the Extreme-Value distribution. The *Erlang* distribution is a good fit for both the single-cell case and for the concatenated-structure case, whereas the *Extreme-Value* distribution does not perform as well overall in the single-cell case.

It is not surprising that the Erlang does so well. It was tried because of the recognition of the similarity of the underlying process from which it derives: that of passing a signal through a set of inter-related transistors in a leaf cell, compared to that of being served by a set of queues in teletraffic systems, as described by Takacs [Tak62]. It has the immediate advantage, by virtue of being effectively truncated at the lower end, of reflecting the fact that there is a fastest operating time for the leaf cell. This makes it a physically more natural candidate for the distribution function than any of the others.

However, any real hope of *significantly* reducing the number of simulations by this method has been removed by these results. It is true that good fits have been obtained after some 30 simulations. But if only 30 simulations are tried then the likelihood of an accidentally-large rogue value occurring is high, and this would surely distort that measures, then the parameters, then the distribution, and finally the accuracy of the percentage of finishing times.

Chapter 8

Analogue Simulation Method

O.K. So this is a mad idea. But it has some interesting consequences!

8.1 Breadboarding a VLSI Circuit

The simulation of MOSFET circuits has gone a long way over recent years into a great deal of mathematical modelling, until, with the advent of practical VLSI systems, there seems to be a retreat from the ultimate sophistication of the very accurate models to simpler, faster, yet hopefully still-adequate models which will handle more nodes and longer real times.

It occurred to me that a fruitful new approach may be possible, based on a fundamental reassessment of the situation.

Consider for a moment the ultimate 'simulation' of a MOS circuit - the circuit itself - that is, in effect, a bread-boarding of the circuit. Of course this is not really feasible (except on multiproject chips for special sub-systems). But what of the next step from this? We can go to equivalent non-calculative methods - after all, a MOSFET itself is the best judge of what its state should be - but failing an actual breadboard circuit, can't perhaps the *SPICE MOSFET* models be used exactly as they are (that is, an arrangement of resistors, diodes, current sources and capacitors) to give the same result? That is, can't they be laid down in silicon? No faster 'simulation' would be

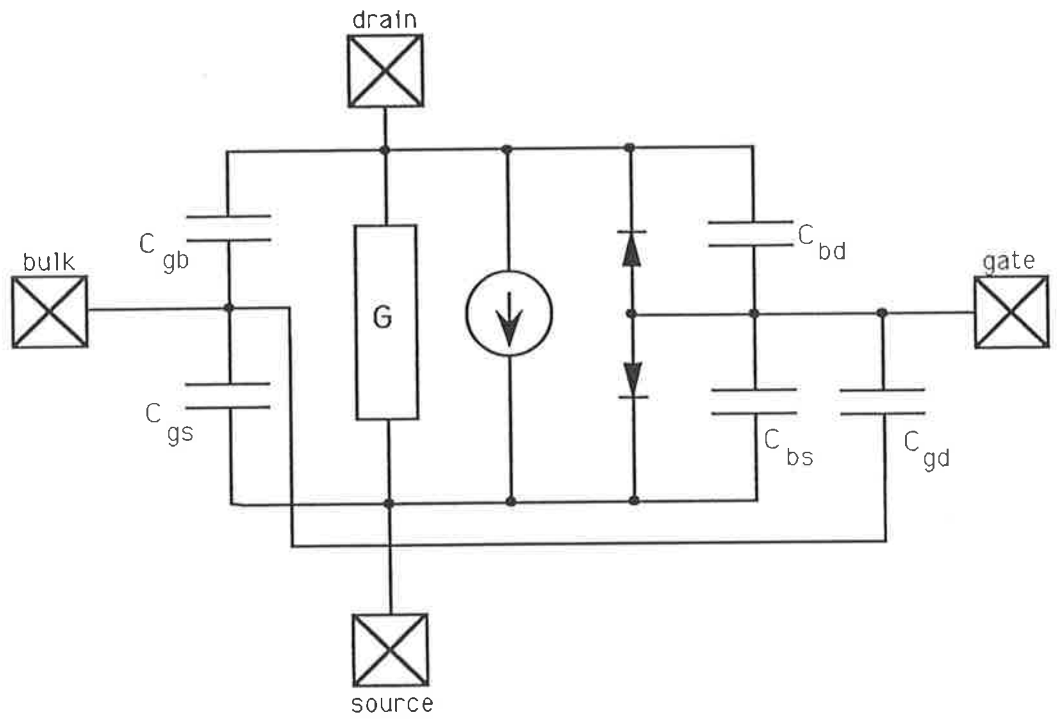


Figure 8.1: Typical MOSFET Model used in Simulation Programs

possible - thousands of simulations could then be run in a few seconds!

This idea determined the next step in this thesis, and, although it ultimately failed, it proved a vital step towards the digital idea described in the next chapter.

8.2 Circuit-element Model

In this non-calculative method I proposed to use a configurable SPICE equivalent circuit MOSFET (an arrangement of resistors, diodes, current sources and capacitors) like that shown in figure 8.1 above, after [ON86].

This was intended as an heuristic step towards creating a system of reconfigurable transistor models on silicon, so that simulations may be done extremely fast in hardware rather than by mathematical models on special-purpose or on general-purpose computers.

The unique feature of this one is the fact that it performs no logical decisions during the simulation, the operations being performed in an inflexible stream; they also perform no mathematics, in the usual sense of the word, to produce the state of the transistors in the circuit. This is equivalent, in the breadboard case, to simply placing actual transistors where they are meant to be - they then behave as only they know how to behave.

8.3 Specification of Analogue Chip

I was so mesmerized by this idea that I designed and fabricated a test structure *via* the Commonwealth Scientific and Industrial Research Organisation AUS-MultiProjectChip/8/83 scheme ['Analogue Test Structure, Author's Publications] before I had fully realised its potentially fatal flaw.

Reference to the plot in figure 8.2 on page 102 shows two inverter structures, identical except for an extra 'always-on' transistor in the right-hand one. Each inverter is composed from the usual simplified equivalent circuit model for an inverter. The capacitive loadings on the input and output lines are switchable to any combination of 0,1,2,4 loads. The extra transistor is to simulate the effect of a different threshold voltage. The Input/Output pad functions are labelled; small probe pads are present so measurements of voltages at input and output can be made.

The inverters have a pullup ratio of 6:1 and pulldown of 1:1 and 1.5:1 respectively. I intended that the switches would be composed of 'native-mode' transistors (doubly ion-planted to achieve almost zero threshold voltage) but this could not be done on this MPC.

The probe pads were for monitoring for various combinations of capacity switched in by the four select lines. An important feature was how much the structure can be slowed down accurately relative to the real operation of such inverters (at their proper scale). Thus comparison could be made with SPICE models.

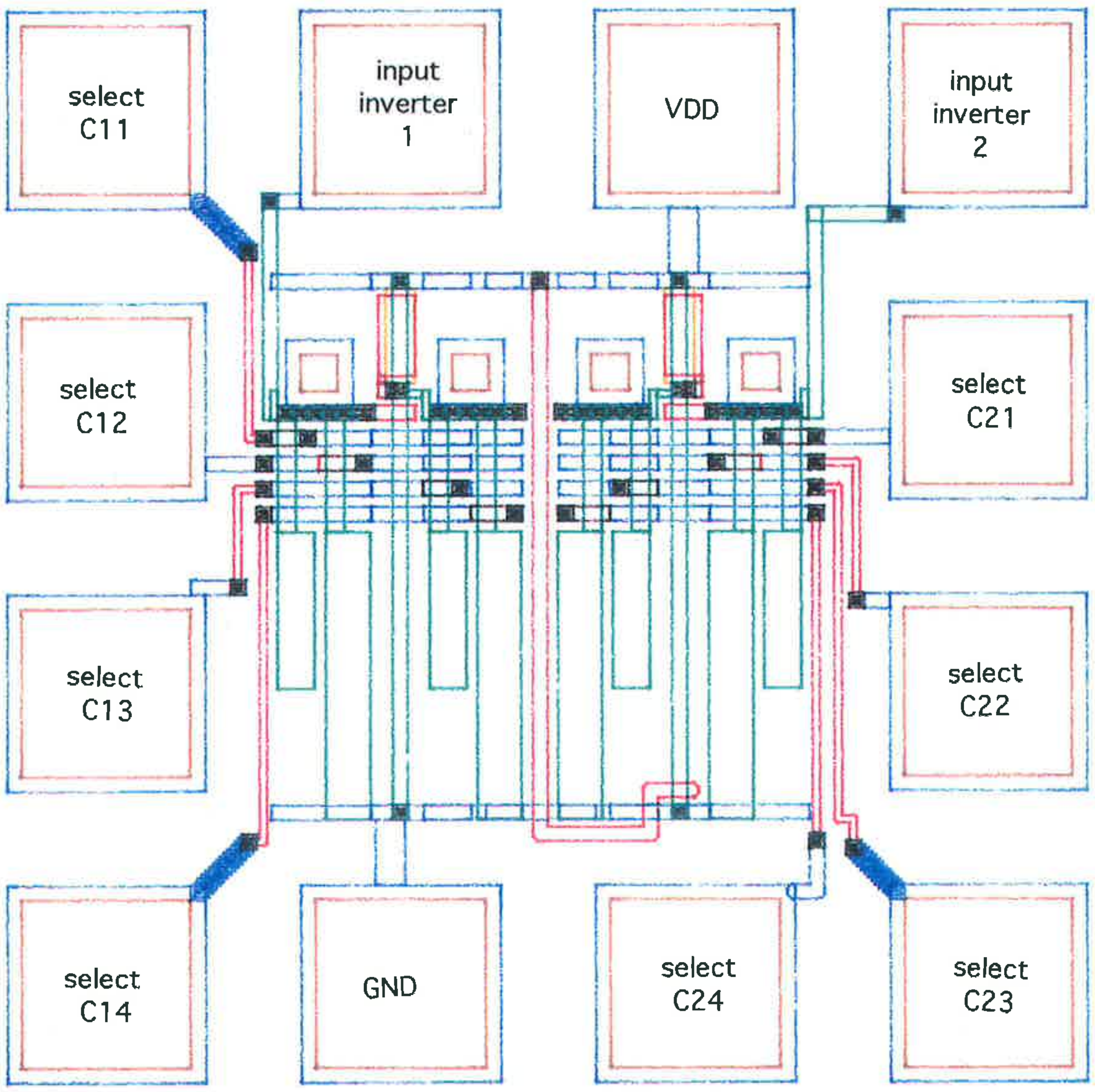


Figure 8.2: Proposed Analogue Array Test Structure

8.4 Fatal Flaws in the Idea

8.4.1 Timing Correspondence

The objections to this are: that the threshold voltage adds to the switched-in voltage; that the transistors run so fast that their operating time and the time taken to pass signals along their interconnecting wires are not to the same scale; and the power rating is wrong. Thus the first requirement is to slow down the models so that the overall system behaves correctly, and to cancel out or minimise the threshold voltage problem.

To ensure that this method is foolproof in providing accurate operation times, configuration C, taking time S in a mainframe simulation, and taking time T on silicon, must map consistently and smoothly. Only in this way can we be sure that doing 150 runs in "silicon space" can map properly onto "mainframe space" - i.e., that the P.D.F. is obtained by "calibrating" the silicon model by doing two mainframe runs, and then doing all the other 150 runs on silicon. I cannot find a foolproof method of doing this.

In addition, Mei and Dutton [MD83] show that there could be problems obtaining enough area for the right ratios of the necessary analogue components.

8.4.2 Flexibility

Another objection to this approach is that the model may not be flexible enough to allow assessment of certain parameters (e.g., body effect) and to allow improvements in the sub-micron area to be incorporated without altering the silicon model; this means that a new chip would need to be designed and made. This would be expensive, and is a fatal drawback.

Chapter 9

Digital System Simulator

Following the failure of the analogue model proposed in the previous chapter, I naturally turn my attention to a digital rendition of the same idea.

9.1 Introduction

It appears worthwhile at this point to examine in detail a method which is close to the natural way transistor arrays operate, which is in accord with the good points urged by Hillis [Hil82]; implements the model of computation, of Smitley and Iobst [SI91]; and which might fit nicely into future system architectures.

From here, there are two broadly discernable directions that might be taken in modelling transistor systems:

- use a few, large, expensive, sophisticated processors, or
- use many, small, cheap, simple processors.

These alternatives have quite different implications for memory arrangement and microcode choice; however, they might after all amount to about the same thing as far as cost is concerned.

However, the class of problem for which one is devised can change when new techniques (sequential or parallel processing), new algorithms (as will be seen), and new architectures (such as neural networks) are devised.

A lot of effort [ACS85] [BD92] [BJS90] [CMS92] [Sch89] [LMP82] [PR82] [SI91] [Vau92] has already gone into distributed processor architectures of this first type. They have produced many very good machines. But their development has meant neglect of natural architectures which it seems, notably by Hillis [Hil82], might now be re-examined with some profit. Recent work by Wilding *et al* [WTHP91] Distante *et al* [DSSSG91] and Schaefer [Sca91] confirm that it is time to re-evaluate this architecture.

How do these alternate specifications: *many, small, simple, cheap*, help determine the architecture?

many these are device-oriented rather than node-oriented.

small if they are small they have little on-chip memory for their own microcode. Thus they have to get their instructions externally. This means that they cannot run as independent processors - they run either as datapath computers or in lockstep.

simple means that they are based on the RISC[1] approach to design;

cheap implying high yields and mass production.

It is known that any given single transistor can only use neighbouring state information in order to act. They do not have physical access to global information. Thus a new model might be sought in the form of a natural array processor, with only nearest-neighbour connections allowed.

The requirement for global information - for one transistor to 'know' the state of another - is the source of intervention by the host workstation in the simulation process. Since all the current accurate timing and analogue simulators use global information, then it is necessary to create a new kind of simulator - what might be termed a <u>contact</u> simulator.

[1]Reduced Instruction Set Computer

9.2 Rationale for Digital System Design

SIMD systems have often been examined previously, but only for very regular algorithms, such as matrix solutions, rather than for circuit simulation in the way proposed here. The success of the MIMD distributed architectures in the circuit simulation field has seen SIMD relatively neglected and left to much of its original applications. However, now that certain MIMD limitations, related to the maturity of the architecture and discussed in Part IV, are becoming more widely appreciated, there might be a new future for an SIMD architecture with many simple processors in circuit simulation after all.

9.3 Aims of Simulation

The aim of these chapters is to determine sufficient detail needed in the *PEs* to be able to write a simulator for the array and answer the questions:

- What is the best convergence algorithm?
- How many bits are needed to produce a given accuracy?
- How long will a simulation take?
- What is the best transistor model?

When this is done the performance of this system can be compared with other methods of achieving the same result. The system simulation in this thesis concentrates on evaluation of convergence algorithm, bits accuracy and time taken.

9.4 Design Considerations

Kung [Kun87] reports that to get maximum speed-up from array processors requires relatively little inter-processor communication. An array of transistor models, running in parallel (concurrently) and testing their nearest circuit neighbours by communicating their state to them, satisfies this requirement in the first instance.

Each *p.e.*, in reality, will calculate its state from neighbouring information and produce new state values to be exchanged.

If each *p.e.* contains its own transistor mathematics, and parameters, then the prime question is: under what conditions is convergence of the solution guaranteed? This is discussed by Nevanlinna [Nev89].

How can the one model be used for two transistors that might be fundamentally different? - e.g. an enhancement nMOS and a depletion nMOS?

How are the transistor models in the processing elements to be laid out on a circuit board? And how will they interconnect?

If a leaf cell is to be "partitioned" into neighbouring transistors, assume that in general each processing element, or *p.e.*, contains a transistor model.

Decisions need to be made about:

- the architecture of the simulator;
- the form of the array processor;
- the independence of processing elements;
- the simulation strategy;
- the partitioning algorithm;
- the mathematical model.
- the solution algorithm

All these points are now examined.

9.5 Simulator Architecture

There are two aspects to the system architecture:

- the choice of decomposition grid;

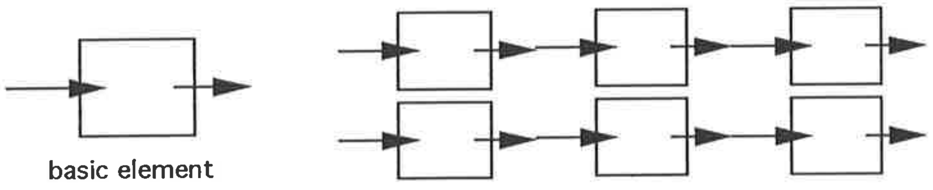
• the PCB layout or equivalent.

9.5.1 Choice of Decomposition Grid

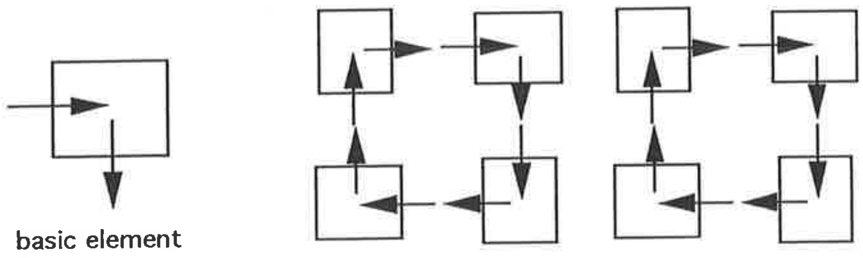
On a grid of processing elements, each of which models a transistor, connections to neighbouring transistors can be made in many different ways (see Cantoni *et al* [CFL91]) but not so that the wires go over an intervening circuit - wires could short out by touching. So one fundamental restriction is that only nearest neighbours connect.

On a normal VLSI wafer, each processing element might be mapped onto the array so that it connects to any number of nearest neighbours; some, after rotational symmetry and mirror-imaging, are shown in figure 9.1 on page 109. Some of these connection schemes are impractical or contain contradictions, because all elements are to act identically and simultaneously,

1 nearest neighbour means that no voltage and current can propagate beyond each two.



2 nearest neighbours means that influences can only propagate in one dimension or in closed loops, as shown below:



3 nearest neighbours means that when they are placed in sequence on the

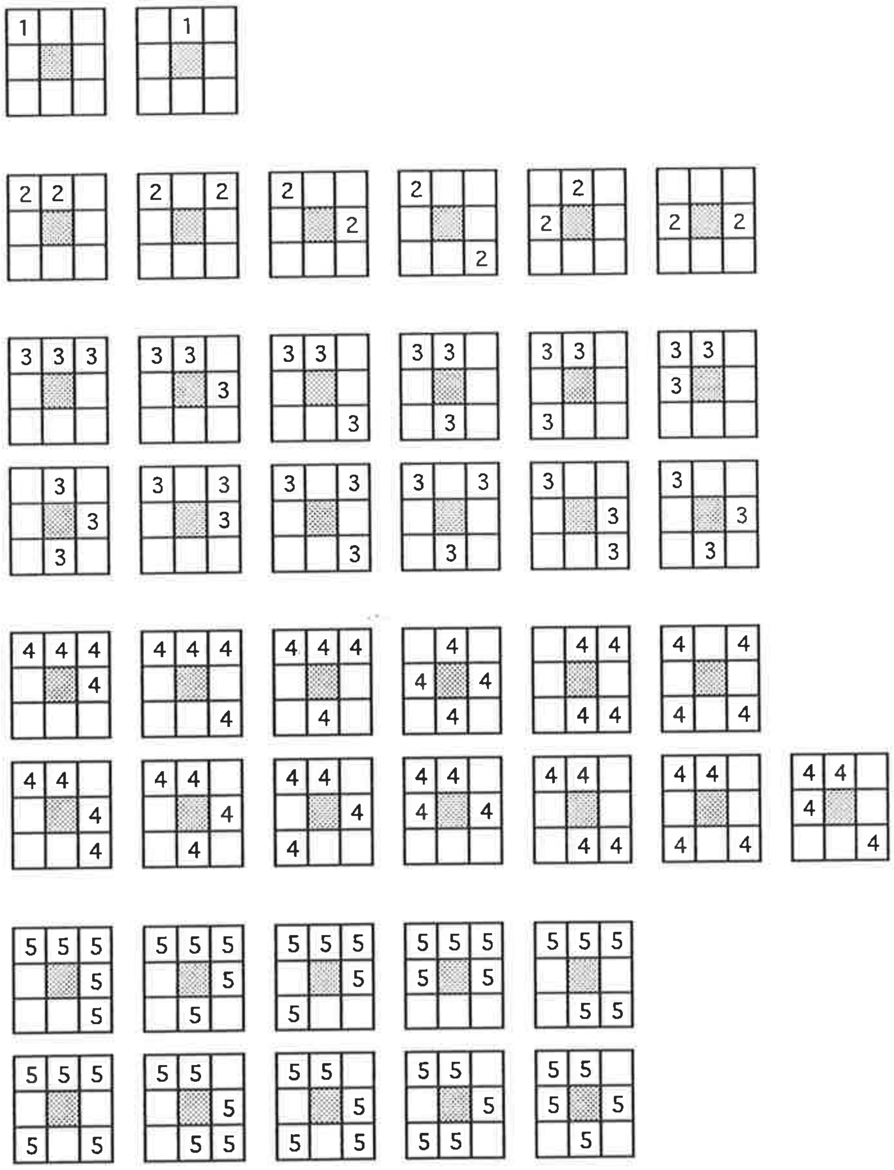
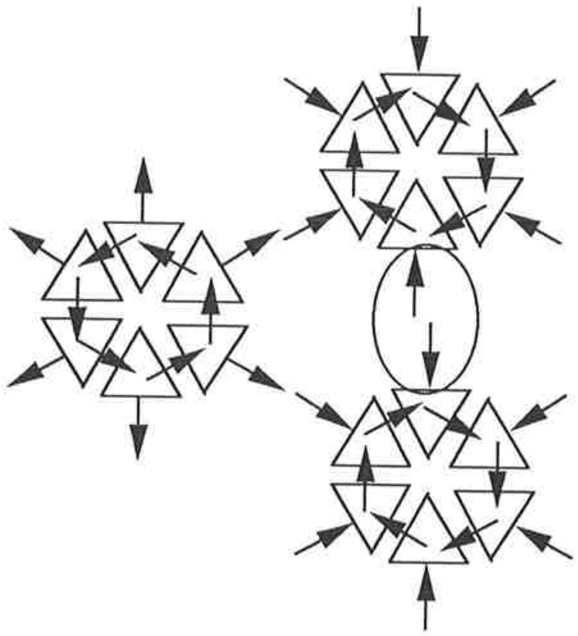


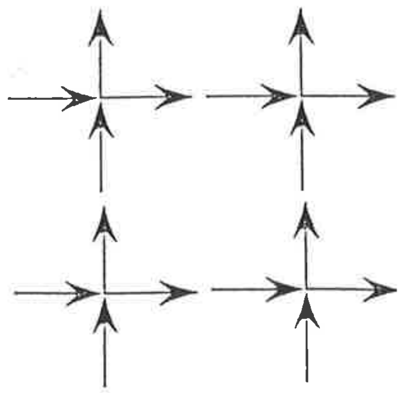
Figure 9.1: Grid of Integrated Circuits or Processing Elements

grid as shown below:



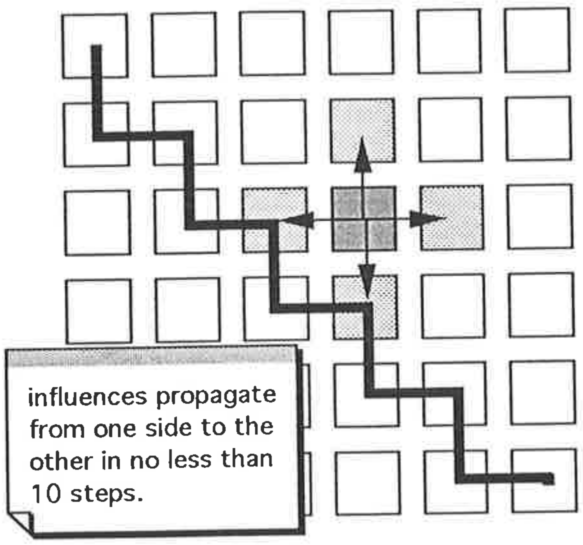
although each set of six can be made to exchange data consistently, separate sets encounter a contradiction *(circled)* when expanded across the grid. So this is an impossible case.

4 nearest neighbours presents no such problems, as illustrated below.

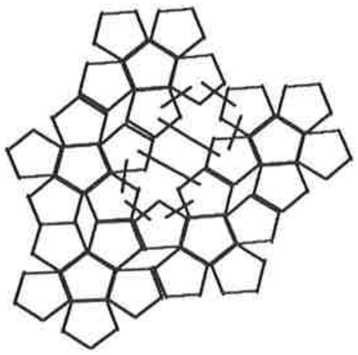


Note that four is the minimum number of neighbours if want the set to be device-based; each p.e. containing a transistor model needs four connections: drain, base, source and substrate.

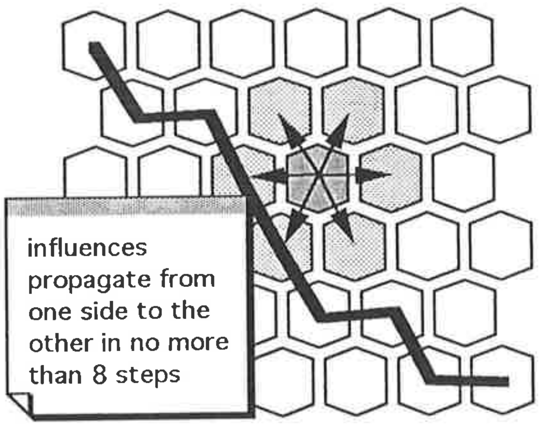
However, with four nearest neighbours, while easiest for initial simulation and requiring little storeage, it takes longer for changes to get to the boundary and back, as shown below.


influences propagate from one side to the other in no less than 10 steps.

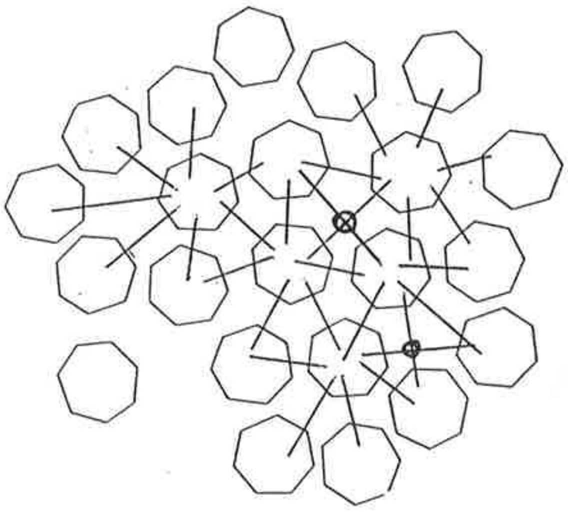
5 nearest neighbours is possible to arrange as shown, but not on a rectangular grid.



6 nearest neighbours presents no consistency problems, but note that six neighbours allow faster propagation of influences than four, as shown.

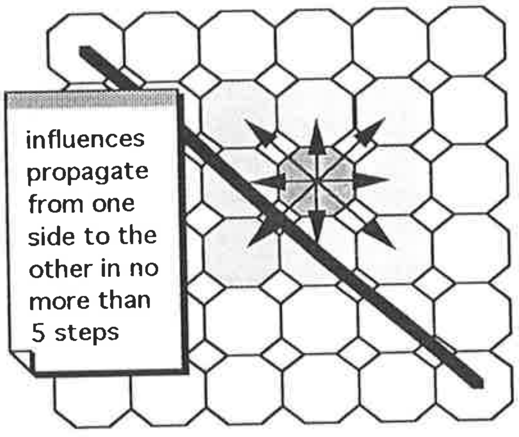

influences propagate from one side to the other in no more than 8 steps

7 nearest neighbours cannot be laid out on a rectangular grid so that they connect consistently, and when laid out as shown



the fatal flaw is exposed at the points *circled* where wires have to cross over to connect 7 nearest neighbours; this is explicitly forbidden in this system.

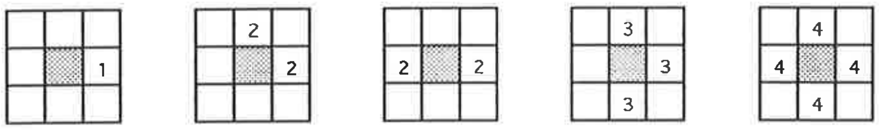
8 nearest neighbours is problem free, like 4 nearest neighbours. In addition, although requiring twice the storeage of four nearest neighbours, eight nearest neighbours allows faster movement of influences to the borders - only five steps per 6 by 6 set, compared with 8. Thus in the long run it cannot be practically written off without an investigation of which influence predominates.


influences propagate from one side to the other in no more than 5 steps

9.6 Form of the Array Processor

In order to make a start on a definite model, this still-numerous set will be restricted so that only manhattan-geometry links are allowed - this eliminates the sets with 5,6,7 and 8 neighbours, since all of these necessarily have diagonal links. However, this does not rule out the possibility that the 5,6 or 8 cases might work quite well; this is discussed in Part 4. The efficacy of these architectures is left to future research and they are not used as the basis for the present heuristic simulation.

This restriction leaves five sets as shown here.



For the reasons explained above, the smallest symmetric set without diagonal links that will connect and communicate properly is 4 nearest neighbours, so it is chosen for the simulation architecture.

9.6.1 PE Layout

According to the arguments of Hillis [Hil82], it makes sense to layout the array so it runs in lockstep as an *SIMD* mesh-connected computer. It is mesh-connected because that is the way real leaf-cells arrange their transistors.

It is *Single Instruction Multiple Data* because each *p.e.* depends only on its 4 nearest neighbours' state and the transistors are all the same and take the same time to solve the equations.

Thus it makes sense to choose a SIMD tesselated computer architecture rather than a datapath or distributed bus. This is shown schematically in figure 9.2 on page 115. The microcode that drives the transistor model is in the top part and is easily altered.

In view of current trends in parallel architectures, the important issues of *pipelining* and *scaleability* are addressed in Part IV.

9.7 Independence of Processing Elements

How can elements, containing different models, run in lockstep? If M_1 is the equation for the first model and M_2 for the second, then a linear combination can be taken

$$\alpha M_1 + \beta M_2$$

with conjugate coefficients (0,1 or 1,0) and this is used in every element; this is quite general.

However, if the *PEs* are required to compute their state in step, and the same, single model is used by all *p.e.*s, then the time taken for the array to agree on the values for the timestep depends only on the slowest *p.e.* - thus any advantage in having faster *p.e.*'s finish early is negated. This is a potential disadvantage of the SIMD method.

The fact that each *p.e.* works in parallel[2] is to be simulated on a sequential machine[3] rather than a parallel one. Thus much care needs to be taken to ensure that values are not overwritten too soon.

It is achieved thus: at each timestep the transistor model is applied in turn to each *p.e.*; then they each exchange any necessary neighbour values

[2]computes its internal state simultaneously with all the others

[3]a SUN/SPARC workstation

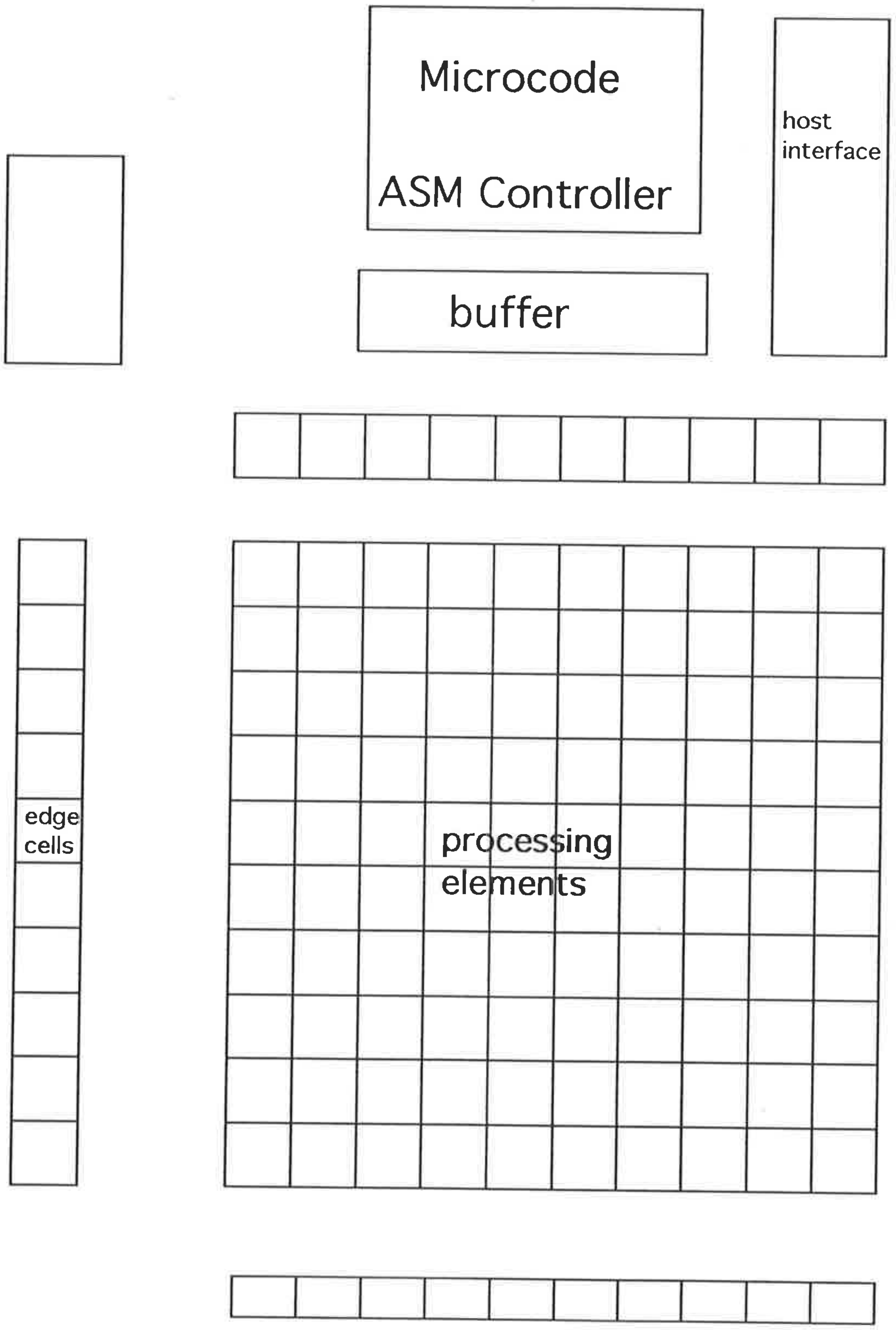
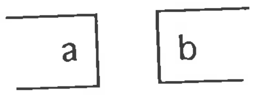


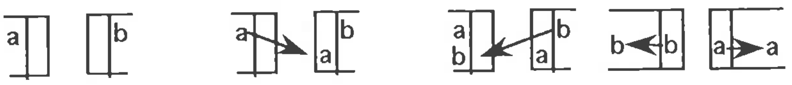
Figure 9.2: Schematic SIMD Computer Layout

(as *p.e.*s); and this step is repeated until convergence.

Consider 2 such exchanging information 'a' and 'b'

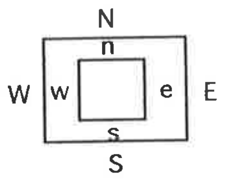


In reality, they will swap these simultaneously. In the current simulation, this is done by the following sequence, conceptually:



This has to be done carefully like this so that the values are not overwritten at an early stage.

In general, then, for every *p.e.*, using compass notation:



- All the elements take in edge values of 4 neighbours into temporary storage;
- Then all elements overwrite the old value with the new value just obtained.

This is a general feature of the simulation programme.

9.8 The Simulation Strategy

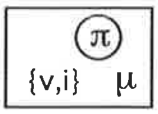
It has seen that the leaf cell is the smallest structure from which it is possible to calculate the systems statistics. Thus this system is designed to carry out circuit simulation of leaf cells. It accepts voltage inputs and produces voltage and current outputs under interrupt control from a host workstation.

In this section the minimum set of definitions and postulates necessary to define the array processor and sufficient to allow a simulator program to be written for assessing its behaviour are set up.

A *leaf cell* is that stage of a modular design of a VLSI circuit at which the function can be conveniently replaced by circuit elements (transistors, wires) that implement the function. It is composed of an array of *processing elements.* If it is designed by regular and hierarchical methods then it already has the right structure for a statistical investigation – otherwise it might not.

A *processing element* is that part of a circuit bordering at most a single transistor. Its function is to simulate the behaviour of a transistor at that place in the circuit.

Real transistors in a leaf cells sense the voltages of its nearest neighbours and produces currents consistent with those voltages. In this digital simulators, a *processing element* must contain or have access to a mathematical model of transistor action, represented by



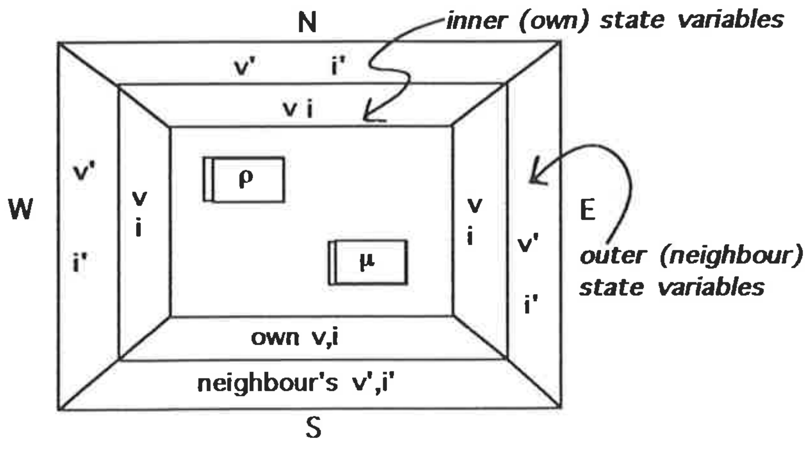
It must contain its own voltages and currents and the parameters ρ of the model:

$\rho =$ **set of sufficient transistor model parameters**

$\mu =$ **model equations**

After calculation of its own currents from its own voltages it needs to adjust to the nearest neighbours by reading and comparing their $\{V,I\}$ set.

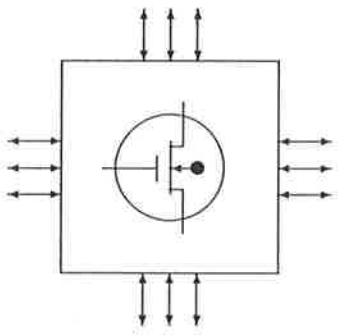
Thus conceptually it now looks like


N
inner (own) state variables
v'
i'
v i
W
v'
i'
v
i
ρ
μ
v
i
v'
i'
E
outer (neighbour) state variables
own v,i
neighbour's v',i'
S

9.9 The Partitioning Algorithm

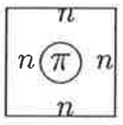
The following nomenclature defines the relevant parts of the simulator.

Processing Element A processing element is the model of that part of a circuit contained within a border drawn around at most a single transistor, topologically connected to its nearest neighbours in the actual circuit. It is represented by the symbol



Array Processor The array processor is conceived to be laid out in the form shown in in figure 9.2 on page 115. In general it is of any dimension necessary to span the circuit transistors.

Canonical Form The canonical form of a processing element is that which has an equal number of ports on each side. If the number of ports is 'n' then it is denoted by π(n) or by the symbol



Any transistor model can be represented by the processing element $\pi(1)$, having only one connection per face to its nearest neighbours. This is because transistors have 4 connections: a gate, drain, source and substrate. The substrate is often connected to the source, however, or can be modelled inside the element. Each processing element contains the mathematical model of at most a single transistor.

Decomposition Every processing element may be decomposed into a larger number of elements containing at most only one transistor.

Structure Theorem A leaf cell of a VLSI circuit may be simulated by mapping its transistors and their connections onto an array processor such that the "nearest neighbour" property is conserved.

This means that the structure of the modeller is quite different from others found in electronics that map sub-sections of the circuit onto a linear set of processors essentially at random.

The mapping leaves open what might be put within a processing element. The view adopted here is that the most appropriate entity to occupy one processing element is one transistor.

These specifications simplify the mathematics inside processing elements, at the expense of their proliferation.

To illustrate this, consider the example of a flip-flop circuit, used throughout these simulations in its *nMOS* realisation, and specified through its various development stages in figure 9.3 on page 120.

To get an idea of the way that processing elements proliferate a rough decomposition of the adder cell, already used as an example of cell characterisation, is shown in figure 9.4 on page 121. This shows that a grid of 121 elements are needed for just 35 transistors.

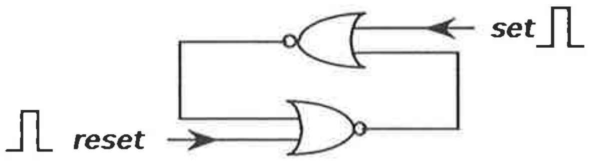
9.10 The Mathematical Model

The system requires access to:

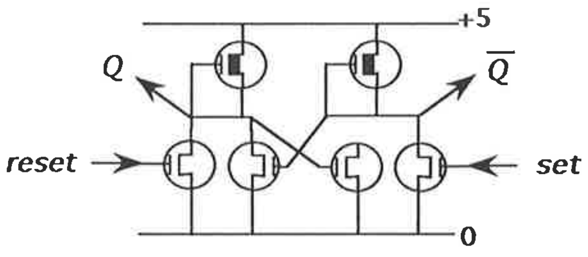
processing element specification i.e., the contents of the *p.e.* as specified by the parameters of the model;

the mathematical model i.e., the equations used in every *p.e.*.

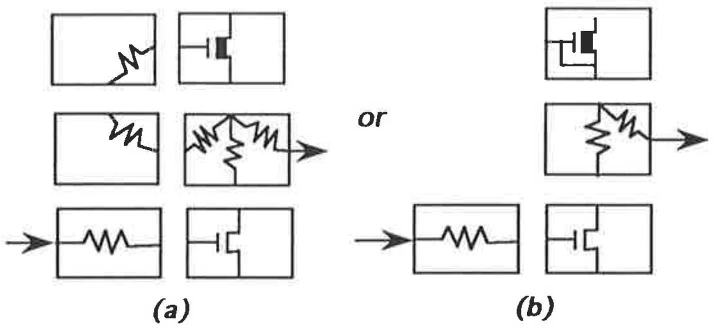
symbolic



implementation



partitioning



grid layout

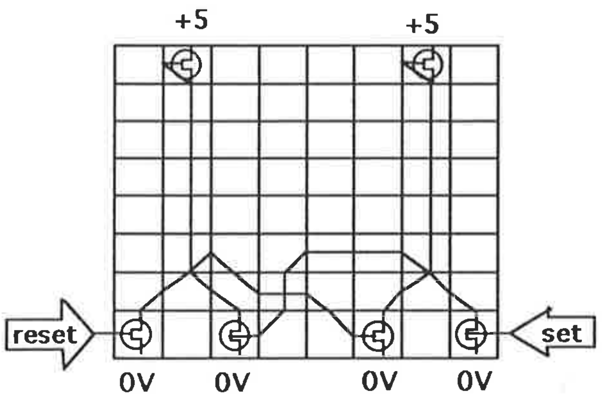


Figure 9.3: STAGES OF PARTITIONING OF FLIP-FLOP CIRÇUIT

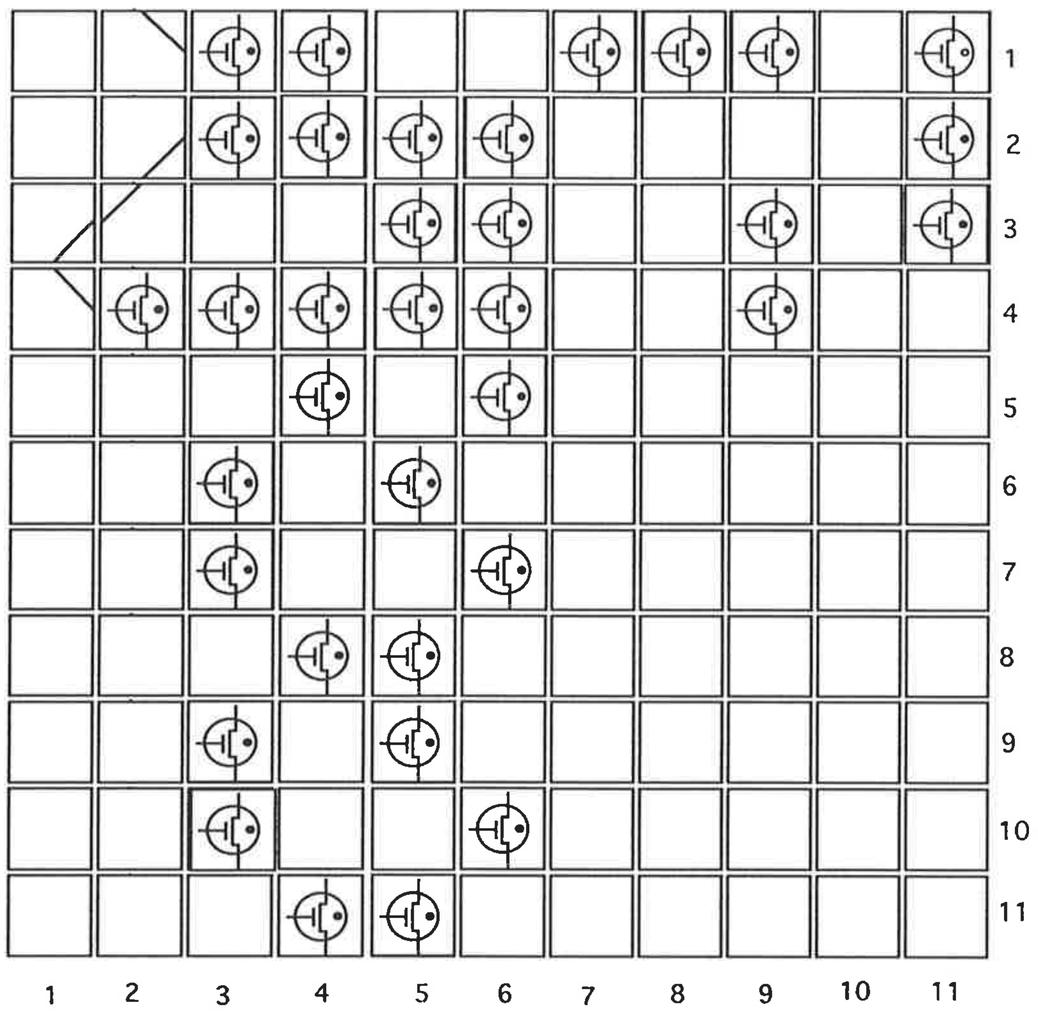
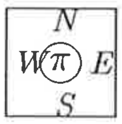


Figure 9.4: Example of Decomposition of Adder Leaf Cell onto Conceptual Array Processor System

9.10.1 Processing Element

Consider the situation of each *processing element* containing either one transistor model or one resistor network, with voltages and currents in each at the east, north, west and south edges, arranged and defined thus:



For the transistor model, a very simple one-equation representation of source-drain current as a function of source and gate voltages is given in Appendix D thus:

$$\begin{aligned}
V_{GS} &= V_{west} - V_{south} \\
V_{DS} &= V_{north} - V_{south} \\
I_{north} &= \beta e^{\alpha(V_{GS}+\delta)}(1 - e^{-\alpha V_{DS}}) \\
I_{south} &= -I_{north} \\
I_{east} &= 0 \\
I_{west} &= 0 \\
V_{substrate} &= 0
\end{aligned}$$

where the parameter $\boldsymbol{\alpha}$ shapes the characteristics, the parameter $\boldsymbol{\beta}$ scales the drain-source current (in mA), and $\boldsymbol{\delta}$ models depletion-mode transistors.

The leaf cell is conceptually divided into many small sections; each section contains a *processing element.* Each has parameters of the model contained in the element, α, β; each has information from the neighbouring elements; there is a common *algorithm* which works on each element in turn, without, of course, destroying the internal data, because really the algorithm is acting on all elements at the same time.

The orientation of the *p.e.* is simply specified by saying which face *(N,E,W,S)* the gate is on; the rest follows the orderly rotation of the basic transistor.

I shall show later that, using my idea of using the opposite voltage and current *as if it was correct for the time being*, that we get very fast convergence, and that this method can easily be shown to have a very sound graphical justification; so it occurs to me, that perhaps, in the case of four nearest neighbours, where there is absolutely no possibility of a graphical visualisation and hence approach to the solution, then I could use the same method - which might also lead to a fast iteration in the more complex case.

9.10.2 Physical Laws

The physical laws of current continuity and voltage continuity imply that at each boundary between elements the voltages are the same and the currents are the same. This follows from Kirchoff's Laws [Chi69], viz:

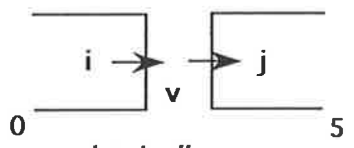
Kirchoff's First Law *that there can be no accumulation of electricity at any nodal point of a network* means that the currents on adjoining faces are equal;

Kirchoff's Second Law *that the electric potential of every point in a network must be single-valued* means that the voltages on adjoining faces are equal.

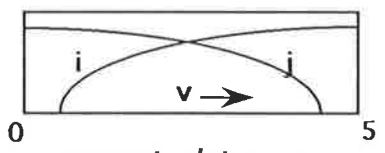
Each element, on each iteration, is required to so adjust its own voltages and currents that they more closely match its neighbour's values. According to Lightner, [Lig87] this does not make Kirchhoff's laws part of the model equation in order to simplify them away; however, it does reverse the approach taken by most simulators, which involves explicit or implicit construction of the Thevenin equivalent circuit. The present method has its compensations, however: it makes it very easy to decide if the true solution is already well approximated and the iteration can be stopped, thus solving one of the main problems in iterative processes [Nev89]; and during the timing simulation a wrong result cannot be produced without notice, one of the major concerns expressed by Dumlugol [DOCM87].

To do this, use is made of the

Uniqueness Theorem *There is only one solution for the voltage and current between two processing elements in a linear array; the proof depends on the fact that the transistor characteristics are positive and strictly-increasing.*



physically...



current plots...

9.10.3 Local Knowledge

The solution algorithm is also constrained by the hypothesis adopted at the outset as the basis for the array simulator,

local knowledge postulate *Each processing element is prevented from knowing what is happening in every other processing element, except the nearest neighbours.*

This is not only by hypothesis, but is the situation in which real transistors find themselves. Each knows only this: that each other element is going to follow the *same* strategy, and that each element knows what the four nearest neighbour voltages and currents are.

Each element knows that it can autonomously alter each of its four voltages (on the four edges), and that it can calculate the effect on each of its four currents (out of the four edges); and that it can tell what it ought to do to move closer to the (i,v) values of its nearest neighbours. But it does not know what its neighbours are going to do at the same time. Of course, it could ask them; but then they, to know what they will be doing, will have to, in turn, ask *their* nearest neighbours, and so on, leading to just the sort of complex information-exchanging situation that we are trying to avoid.

9.10.4 Nearest Neighbours

The idea that the circuit may be modelled by mapping it exactly leads to a fundamental restriction not present in the mapped models already described.

If each transistor is to sense only its nearest neighbours, then each one can have *no global knowledge of any kind* - indeed, no knowledge of even what is happening on just the other side of its neighbour, since this is ultimately global.

Thus these two conditions apply:

Transistor Model *Each processing element contains a model of transistor operation sufficient to produce the currents through its four faces given the voltages on its four faces;* that is:

$$i_N = i_N(v_N, v_E, v_W, v_S);$$

$$i_E = i_E(v_N, v_E, v_W, v_S);$$

$$i_W = i_W(v_N, v_E, v_W, v_S);$$

$$i_S = i_S(v_N, v_E, v_W, v_S);$$

Information Access *Each processing element has access to only that state information that its nearest neighbours can provide; i.e. - in practice, only the voltage and current at the common interface between elements. Absolutely everything else, explicitly or implicitly, is <u>unavailable</u>.*

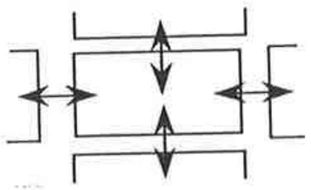
This postulate encapsulates the vital difference between this method and all previous ones. If the system could not be made to work under this restriction, then we would have had to fall back to one of the random distributed-processing models with all the expense and complication that they entail.

The postulate means that we only have to focus on one processing element - which we shall now do.

9.10.5 'Phantom' Edge Elements

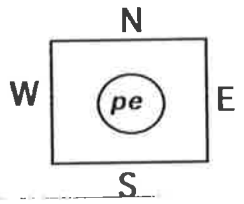
Each processing element has four nearest neighbours, except along the edge of the array. To treat edge cells identically with interior cells in all algorithms, it is sufficient to create a set of 'phantom' elements. These are situated all around the array. Their function is to provide any driving inputs to the cell, and to allow the same algorithm to be applied to all processing elements without exception - since they are part of an SIMD machine - particularly where they an element has to receive data from a really non- existent edge cell.

Thus in general we are always dealing with:

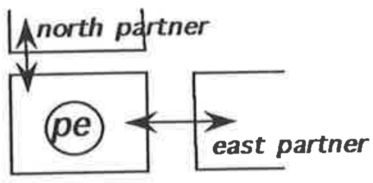


Dealing with even four neighbours at once is even not necessary algorithmically; it can be cut to two by these definitions:

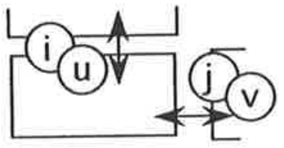
Face Each processing element has four faces labelled N,E,W,S for north, east, west and south.



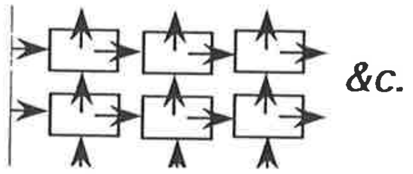
Partner Each processing element has two partners - those to its North and East.



Now, if each processing element only has to consider its voltage and current relationships with its partners,



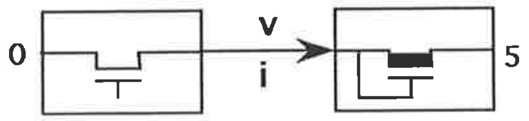
this allows each processing element's South and West faces to be dealt with by other processing elements – for which they are really North and East faces – according to the diagram:



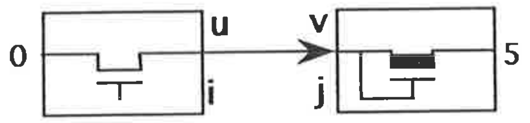
9.10.6 The 'Partner' Algorithm

By the fundamental postulate, each processing element can only 'see' the state variables at the 'face' between itself and the next one.

For definiteness, consider two transistors (as perhaps in a nMOS invertor structure) connected source to drain as in the diagram.



Regarding, for the present, the voltages across this composite structure as fixed at 0 and 5 volts, there is a common voltage v at the face between them which in the quasi-static approximation pursued here, allows a unique current i to flow across the face. No other voltage results in this current. Now consider that each processing element has its own idea, however obtained, of what the voltage on that face should be at this timestep, *i.e.,* consider the situation as if it were:



The left *p.e.* considers u to be the correct voltage at the face, producing a current i through its west face; whereas at the same time the right *p.e.*

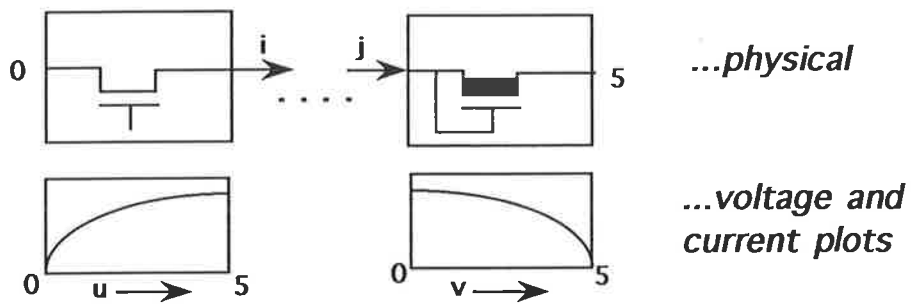
considers v to be the correct voltage, producing a current j through its east face (which is the west face of the other p.e). If the currents i and j differ, then, since this is physically impossible here, one or both of them are wrong in this assessment . . .

9.11 The Solution Algorithm

Since neither *processing element* has any ground for supposing that *it* possesses a more accurate estimate of the common face voltage, then an ingenious method of coming to an agreement emerges, which, as it turns out, is a more general statement of the venerable and powerful Newton-Raphson iteration technique for finding the roots of a function – no wonder it works so well!

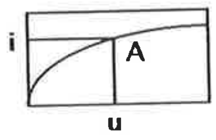
The idea is this: since neither *processing element* is in a privileged position concerning the estimate of the voltage, then imagine that each one, for the time being, (as it were, for argument's sake) agreed to consider, just temporarily, that the other processing element had in fact got hold of the correct value of inter-*p.e.* face voltage, and proceeds to calculate what its own face current would be if this were so.

Consider the diagrams below:

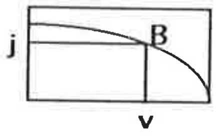


On the left is a plot of the typical current in the left transistor; on the right is a plot of the typical current in the right transistor.

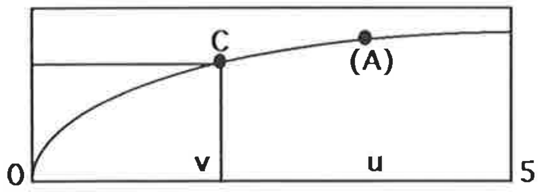
If the left *p.e.* is correct in its assessment of the face voltage, then we are at the point A in



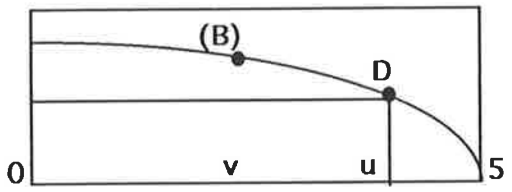
whereas if the right *p.e.* is correct we are at B in



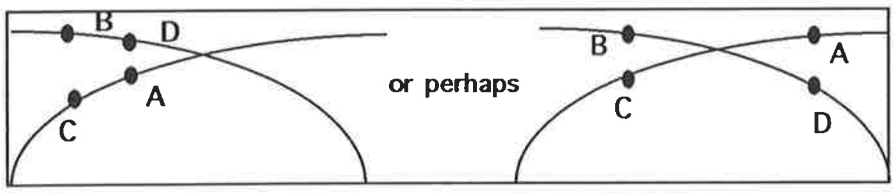
If, by my suggestion, we allow the left *p.e.* to try the voltage supposed by the right *p.e.*, we get the point C in



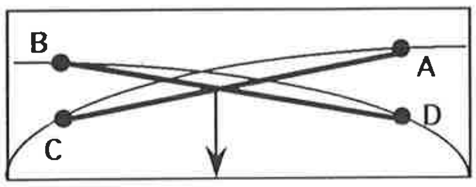
whereas if we allow the right *p.e.* to try the voltage supposed by the left *p.e.*, we get the point D in



Now consider these graphs superimposed on the same scale

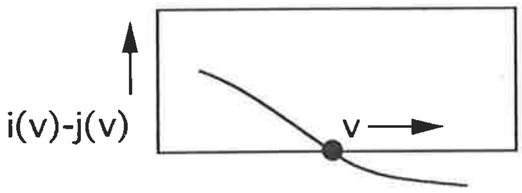


which allows it to easily be seen that rather than averaging the mismatched voltages and currents, a better guess for the solution would simply involve the point of intersection of the lines BD and AC in



This idea, tried out on transistor models joined in this fashion proved that in all cases, and no matter where the starting values of u and v, the system converges to very high accuracy within four iterations of this procedure.

This rapid and strong convergence is surprising at first. It was only later realised that the procedure was practically equivalent to the Newton-Raphson method for finding the root of a function of one variable - in this case, the difference in currents calculated by each *p.e.*, which is if the form



and thus the excellent behaviour of the algorithm was explained. The mathematical reason for this is shown in Appendix C.

An Aside...

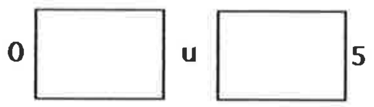
Now, this must look like a silly thing to do - to fail to immediately recognise and use Newton's method to solve the currents between the processing elements. And, indeed, if the primary concentration is on the *nodes* of the array then it would be.

But the point of view motivating this simulator concentrates rather on *devices*. This is a subtle difference which turns out to have profound consequences[4], so in addition to being a silly oversight it also turns out to be instrumental in opening up a new paradigm for the array processors of the future. This is taken up in Part IV in the Evaluation.

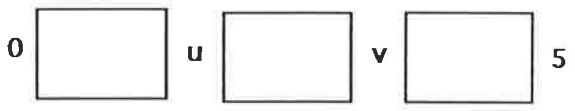
[4]explored in the Game Theory chapter 11, later

9.11.1 Conclusion

Thus is solved the simple case of 2 linear *p.e.*'s. with fixed end voltages.



The next simplest case is somewhat intimidating.



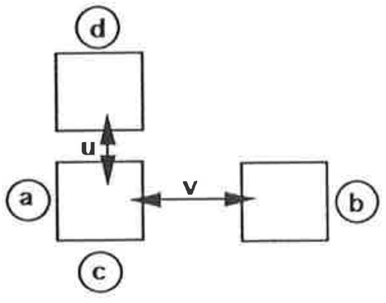
The first two *p.e.*'s. can very quickly (as above) come to a solution for the voltage u - but the value of u, reached by this method, depends on the value of v, which is being determined at the same time, by the same method . . . depending on u. So there is a potentially intricate and implicit parallel determination which threatens to demand global information, thus destroying the postulate on which this approach is based.

The simplest way out of this impasse is to solve for the currents as if there are two sets of independent pairs of elements. This tactic works, as will be seen.

After all, we already have a technique that works for the case that we can visualise; we know it is equivalent to a powerful and well-studied iteration algorithm; so why not leave it to its own devices, i.e., let it have its head at the level at which it is known to produce the correct result?

9.12 Global Strategy with Local Information

In short, my hypothesis, to be judged by its results, is to let each *p.e.* solve its voltage/current relation with each neighbour, pairwise, in turn, without considering what the other *p.e.*'s. are doing at the same time. When that is done they can each take a look at the results of these individual calculations and redo them if the result is too different from previous ones. For example, each *p.e.*



comes to an agreement, by this algorithm described above, on the value of u with its Northern partner and, separately, on the value of v with its Eastern partner, all the while using fixed values of end voltages at a, b, c, d. (The west and south faces are taken care of by other *p.e.*'s. to the west and south, at the same time).

Four iterations are done (from previous 'pair' experiments this is sufficient); each *p.e.* signals that it is 'done' or 'not done' to its North and East partners, these signals being ultimately collected by an AND process out at the top right hand corner of the array (like the global-or line in the CM-2 [SI91]); and the process is repeated until convergence, if and when it occurs.

The next two chapters report what happens.

Chapter 10

Bargain Method

In which I try out my new proposal for very fast analogue timing simulation of microelectronic systems by mapping transistors onto an array of digital mathematical models of transistor operation, on the hypothesis that the bargaining analogy is sound.

10.1 The Array-Processing Model

The model of a circuit discussed in the previous chapter has been studied by implementing it as a C programme run under the UNIX operating system on a Sun SPARC workstation.

The results of that study, broadly speaking, are:

- such a model would work, fast enough to be interactive in the sense already described for programmes of this nature and scope;
- for bistable circuits with feedback, some very general indication of the initial conditions produces a solution- otherwise it takes a very long time to converge;
- for straight-forward circuits of the combinational type, the solution can be started from any initial condition;

- it does not appear to converge in this simple unimproved form for circuits containing pass transistors.

A detailed examination of the process by which this processor-array idea has been assessed is presented in the following sections.

10.2 Results

The embodiment of this approach to array-processor simulation has been realised in the computer programme described in Appendix D. All of the technical and mathematical considerations and definitions necessary to get it working are described there. A summary is given here concerning its behaviour as a simulator of integrated circuits. The results of investigations into the array-modeller are very encouraging, and are presented in graphical form in the following pages.

10.2.1 Convergence

Circuit behaviour has been found and characterised as:

weak and oscillating convergence for **feedback systems** unless the initial state is unambiguously specified - then it is strong; rapid and strong convergence for **non-feedback systems**.

10.2.2 Circuits Used

Initial encouraging results were obtained on simple gates such as inverter rings and NOR gates.

All the detailed simulations reported here are carried out on a feedback

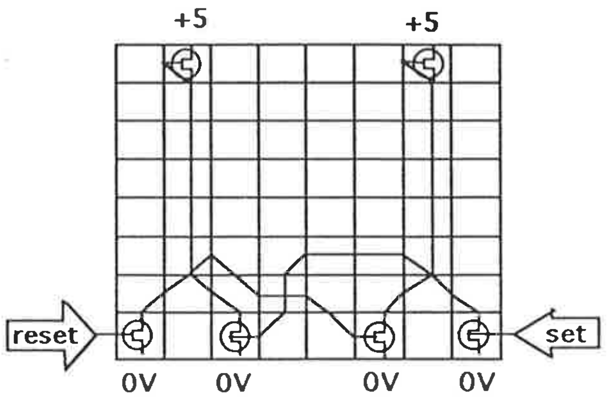
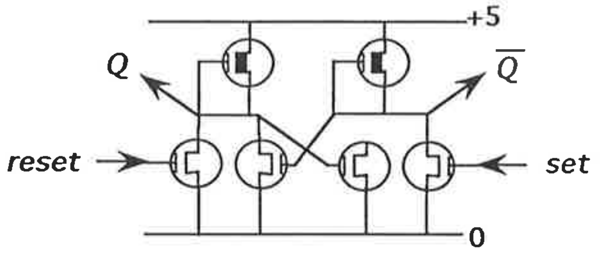


Figure 10.1: ARRANGEMENT OF FLIP-FLOP CIRCUIT ON GRID

circuit comprising a set-reset flip-flop, shown here.



It was chosen because it has a feedback path and is bistable, so is a non-trivial circuit, in the sense that if the algorithms do not work on this they probably will not work on any circuit.

The circuit is decomposed on an 8×8 grid of processing elements as shown in figure 10.1 on page 135.

The simulator produced the correct behaviour through the full set and reset cycle. However it took over 2000 iterations when started from zero[1], as presaged by its early behaviour in figure 10.2 on page 136, whereas it was much faster when the bistable nature of the circuit was unambiguously removed. It was sufficient to indicate a '0' or '3' volt level at the start. The set of startup voltages used for all these simulations is shown in figure 10.3 on page 136. The set and reset signals are pulsed in sequence, and this full simulated sequence of voltages is shown in figure 10.4 on page 137.

[1] *i.e.* there was no indication of the final state - 'flip' or 'flop'

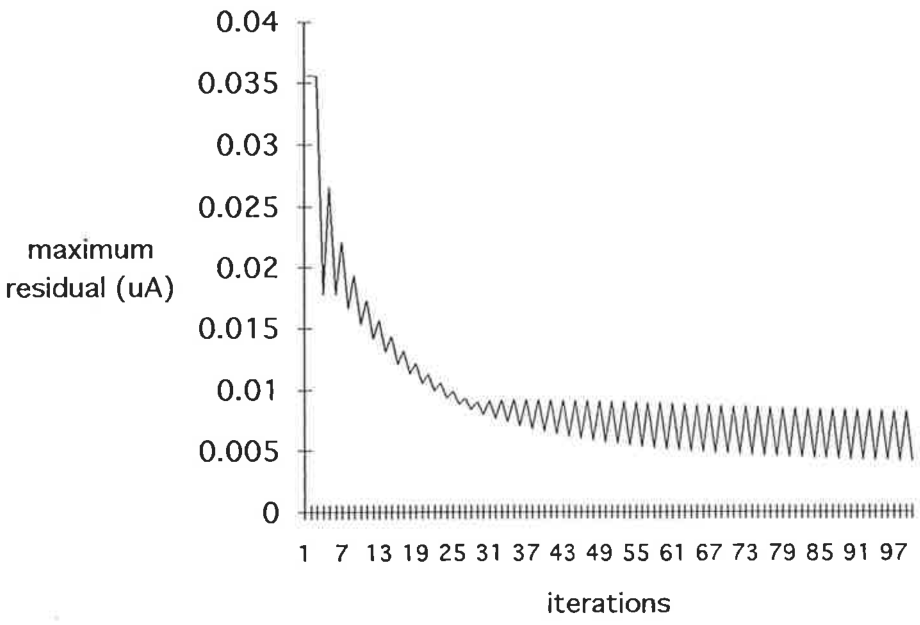


Figure 10.2: Maximum Residual Oscillations in Array when started up from All-Zero Voltages

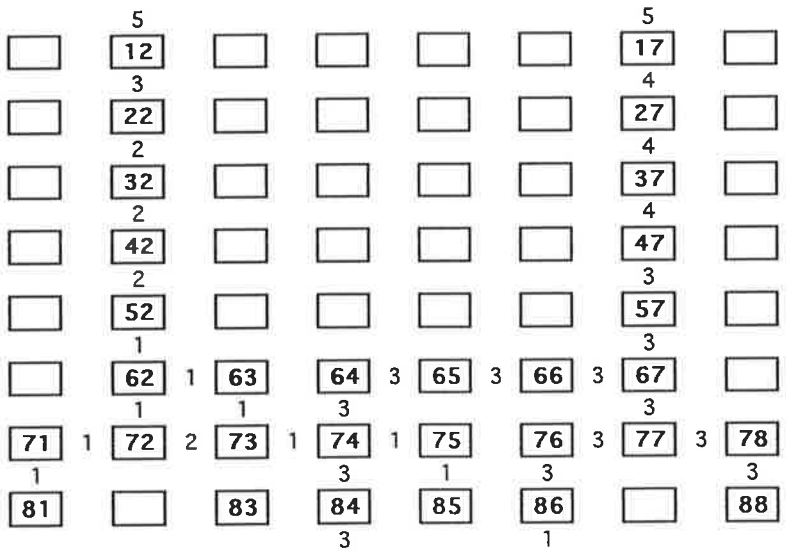


Figure 10.3: Non-zero Startup Voltages and Elements

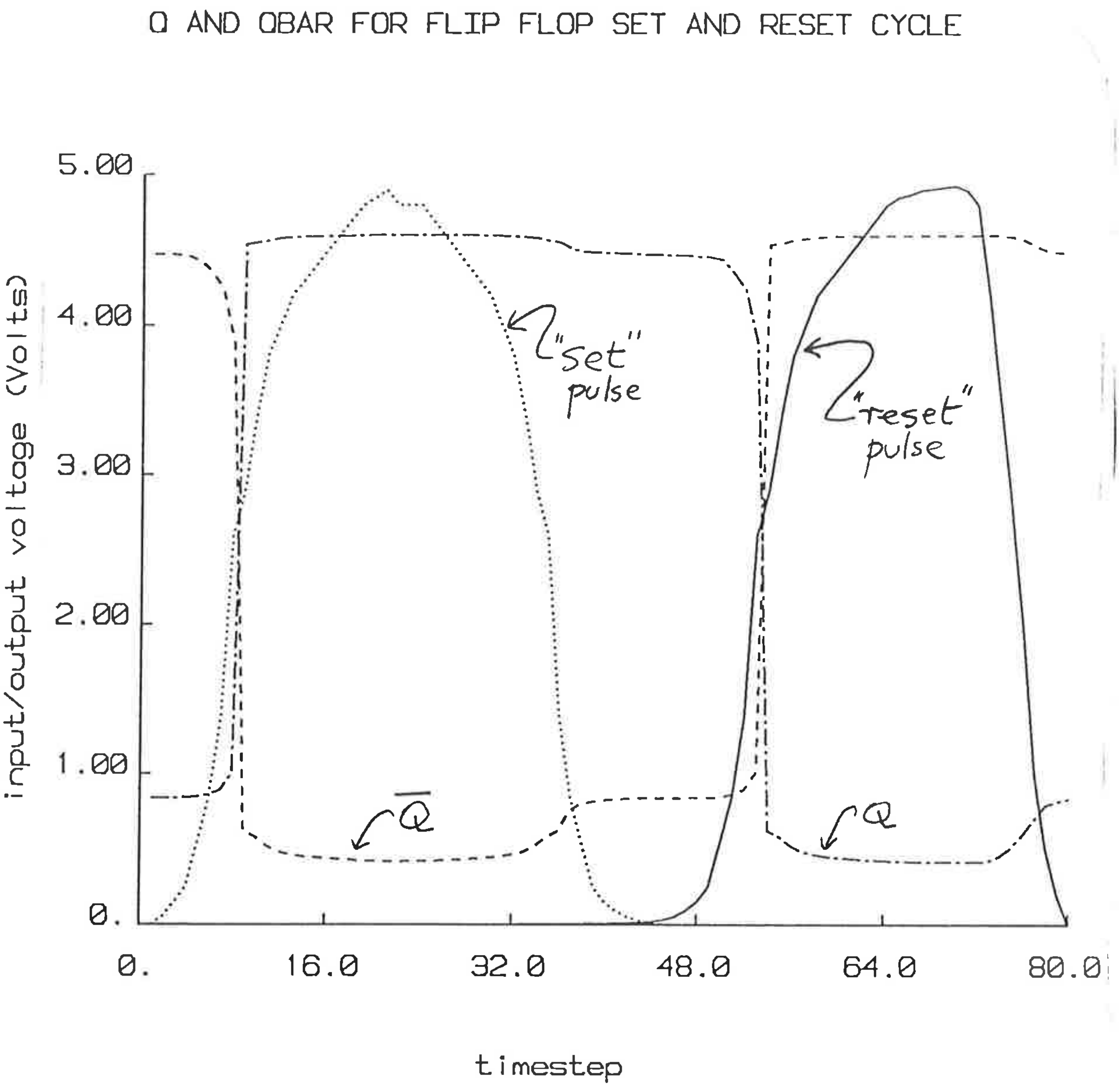


Figure 10.4: Simulation of Flip Flop Set/Reset Cycle

10.2.3 Rate of Convergence

It is shown in this chapter that the convergence rate - how many iterations are needed until the voltages are close to the final result - varied widely and was greater when the input drivers were changing fastest. This is to be expected, since the nodes are most out of balance at those times.

However, it is found that the system has to be quite close to the final state if not too many iterations are needed. If it is just a small way from it then the iterations rise dramatically. Once the system is a long way from the solution it gets 'saturated' in the sense that the number of iterations does not increase much at all. These results are shown in figure 10.5 on page 139.

This result suggests that any improvement in convergence speed should be sought at those parts of the overall waveform where most iterations occur. This is seen to be where the voltages at all nodes are most out of balance, either because of rapid driving-voltage change, or at startup where the initial voltages are not yet settled and an approximate guess must be made. Thus this investigation now concentrates on the startup point in the waveform, shown in figure 10.5 on page 139 to need just under 1800 iterations.

10.2.4 Number of Elements

The results of simulation of rings of oscillators also show that there is a slightly more than linear increase in the time taken to converge as the size of the grid of the array, that is the number of possible processing elements along each side, increases. This reflects the fact that influences propagate in both directions; it is surprising that the rate of increase is so slight, really. These results are shown in figure 10.6 on page 140.

10.2.5 Accuracy

The simulator was run finally with varying numbers of bits in the mantissa by masking all real number operations. Double-precision real numbers in the

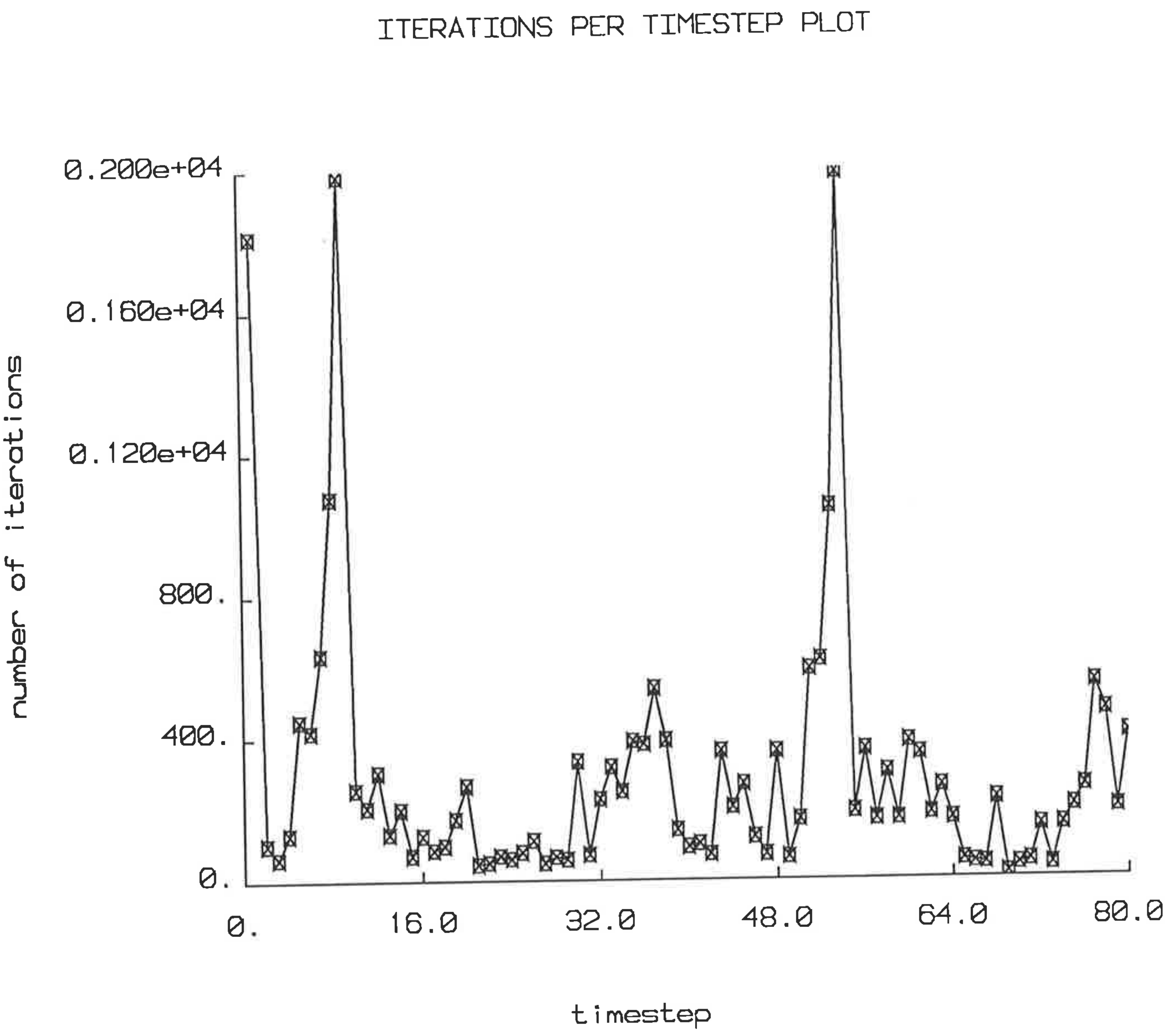


Figure 10.5: ITERATIONS NEEDED FOR CONVERGENCE

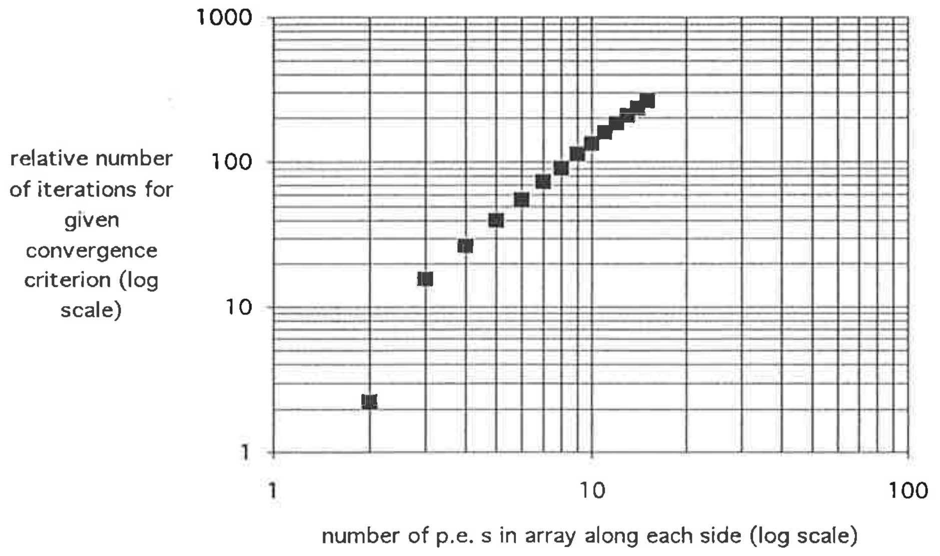
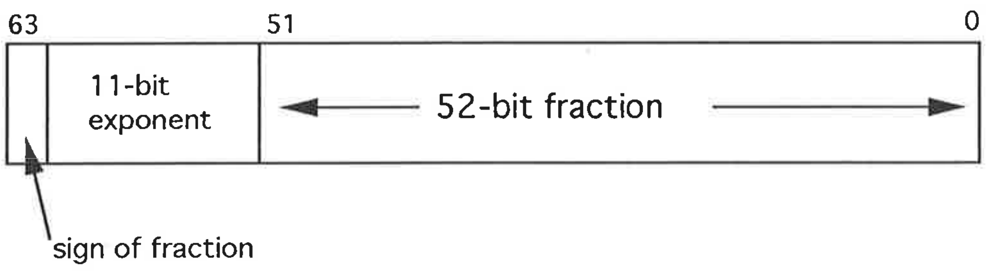
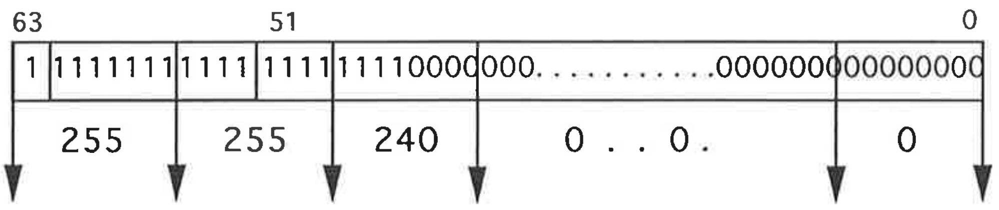


Figure 10.6: Iterations Needed as Processing Elements Increase

SPARC processor conform to the IEEE format[2]. They are stored in eight successive bytes in the form:



so that, for example, if the case of 8 mantissa bits accuracy is being investigated, then the set of byte masks is:



The full set of masks[3] is shown in the C program in Appendix E.

The exponential function is implemented to high accuracy by using the

[2]IEEE Standard for Binary Floating-Point Arithmetic

[3]for 1 up to 53 bits

simple arithmetic operations $+ - * /$: e^x is expressed as $1/e^{-x}$, and where $x > 1$ it is successively halved until $x < 1$; then the series solution

$$e^{-x} = 1 - x + x^2/2!...(-x)^n/n!$$

is used since its maximum error is known to be less than the last term. Finally the result is multiplied by itself the same number of times that it was originally divided, i.e., effectively computing e^x from

$$(e^{x/m})^m.$$

10.3 Newton-Raphson Iteration

The criterion used in this implementation, by default, is that when then four face voltages match exactly,the mismatch between the four face currents vanishes. Mathematically, the measure of the mismatch is

$$(i_n - i_{n0})^2 + (i_n - i_{n0})^2 + (i_n - i_{n0})^2 + (i_n - i_{n0})^2$$

and for this to vanish it is trivially done by choosing the currents to be equal. This is what would be done here by the Newton-Raphson iteration if it was taken to convergence, which it is not.

The Newton-Raphson iteration technique described in general in Appendix C was used in these solutions. It was found that in all cases, and no matter where the starting values of the initial voltages, that the system converged to very high accuracy within four iterations of this procedure. Because the number of mantissa bits is an important parameter in this study, the effect of the number of mantissa bits on this convergence behaviour was investigated and is shown in figure 10.7 on page 142.

Even down to 8 bits the convergence in no more than 4 iterations is observed. This rapid and strong convergence is not even usually used in simulations. It is normal [NSV84] to just obtain the first iteration and use that as the starting point. In these simulations it is found that this is a sound procedure, making less than one percent difference overall to the number of iterations needed.

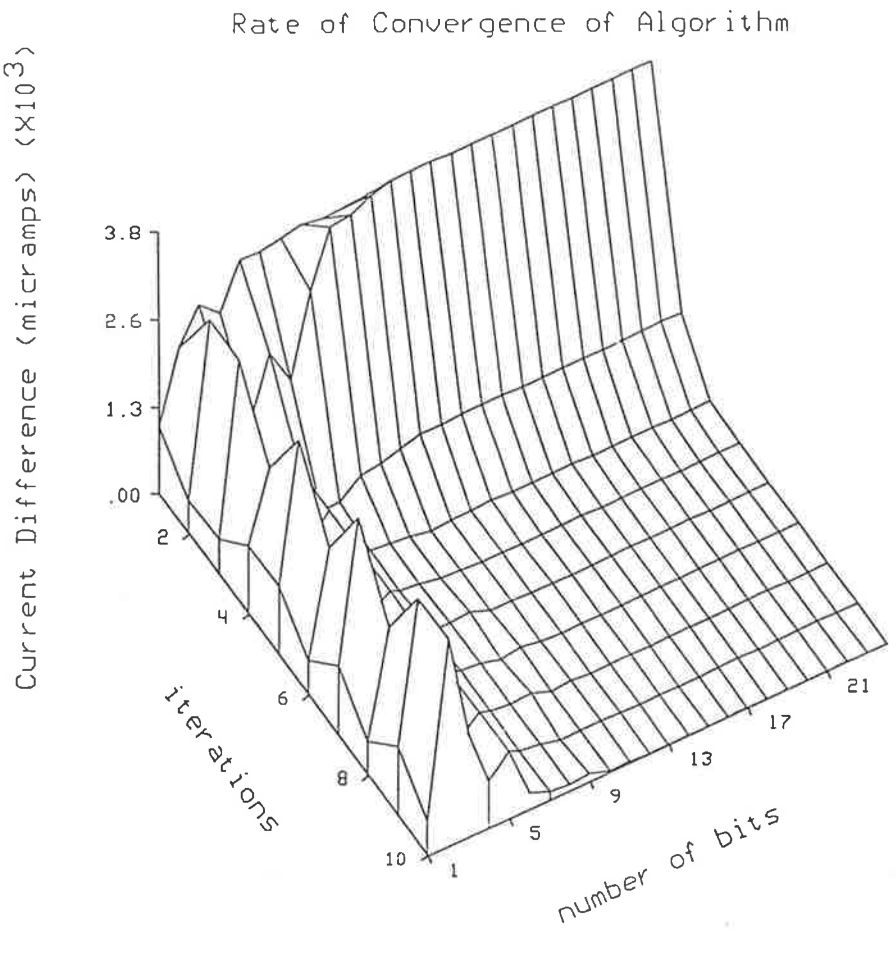


Figure 10.7: RATE OF CONVERGENCE OF NR WITH MANTISSA BITS

10.3.1 Overview of Behaviour

To show what can be achieved, the proposed system is started up from a given initial set of node voltages. Starting up from all zero nodes leads to an arbitrary situation and very long iterations (in excess of 2000), so startup is from a rough approximation to the initially expected state shown in figure 10.3 on page 136.

It is perhaps astonishing that it converges at all. However, it is the purpose of this study to verify that there is any validity the economic analogies that this approach is based on, and because real-life bargaining analogues converge somewhat roughly, this provides an initial measure of confidence that this array must, too.

The criterion for convergence can be either one of:

- the present current and voltage on the face differs negligibly from the previous current and voltage on the face;
- all over the model array the voltages match and produce currents such that the maximum mismatch is smaller than a specified amount.

The second criterion is chosen because it is similar to the familiar relaxation method criterion [Sou43]. The case started is with the maximum number of mantissa bits (53) initially, and 1μA convergence criterion, which is quite tight. Starting from the voltages on the nodes already given in figure 10.3 on page 136, the voltages on the faces N72 and N77 are traced and the maximum mismatch of currents is recorded.

Next examined are three important extensions to this *bargain method* which are: the *smoothing method*, where this zig-zag is largely suppressed; the *look-ahead algorithm*, which gives some speedup; and the *profit criterion*, which enters the new realm of *game theory*, explored in the next chapter.

The convergence properties of the four methods are show in figure 10.8 on page 144, where the *unimproved bargain method* is the upper curve and the various improvements make up the lower three plots. Typical startup

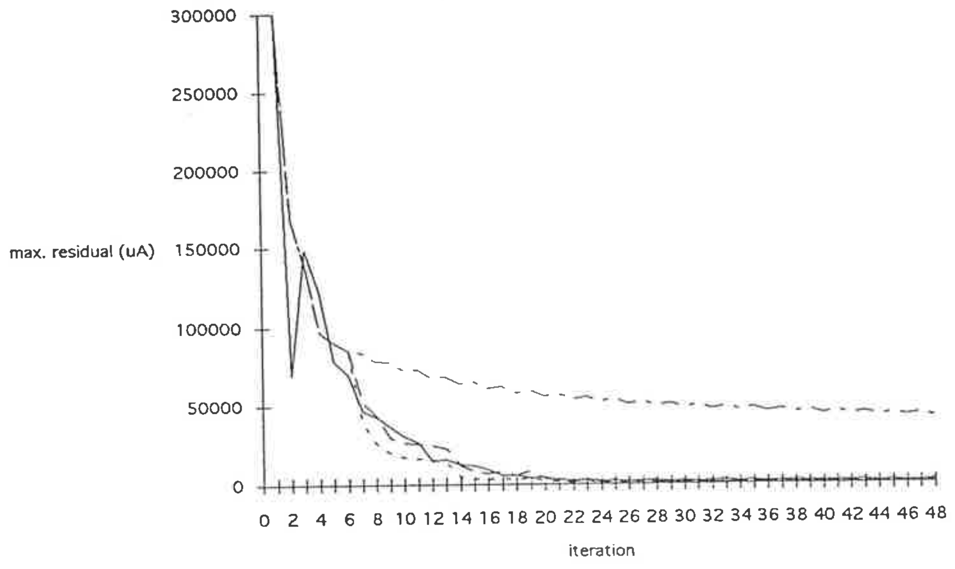


Figure 10.8: Comparison of maximum residuals for all methods

voltage waveforms are shown in figure 10.9 on page 145, where this time the unimproved voltage is the zig-zag one.

Briefly, indicative numerical results for 53-bit mantissa, 1μA convergence criterion, are summarized in the following table:

Method	Iterations	RMS Dist.	Zero Start
pure bargain	1784 iterations	0.014mV	2183 iterations
bargain plus smoothing	965 iterations	0.133mV	2144 iterations
bargain plus smoothing plus lookahead	602 iterations	0.0134mV	1918 iterations
bargain plus profit criterion	966 iterations	0.0114mV	2206 iterations

where: the RMS Dist. is the root mean square distance of all the startup voltage nodes from all the voltage nodes of the true final solution; and Zero Start means that if the system is started up from all-zero nodes, it comes up into the 'flip' stage rather than the 'flop' state, and takes somewhat longer.

In addition, indicative numerical results for 53-bit, 1000μA convergence criterion, including the *relaxation method* results for completeness, are summarized in the following table:

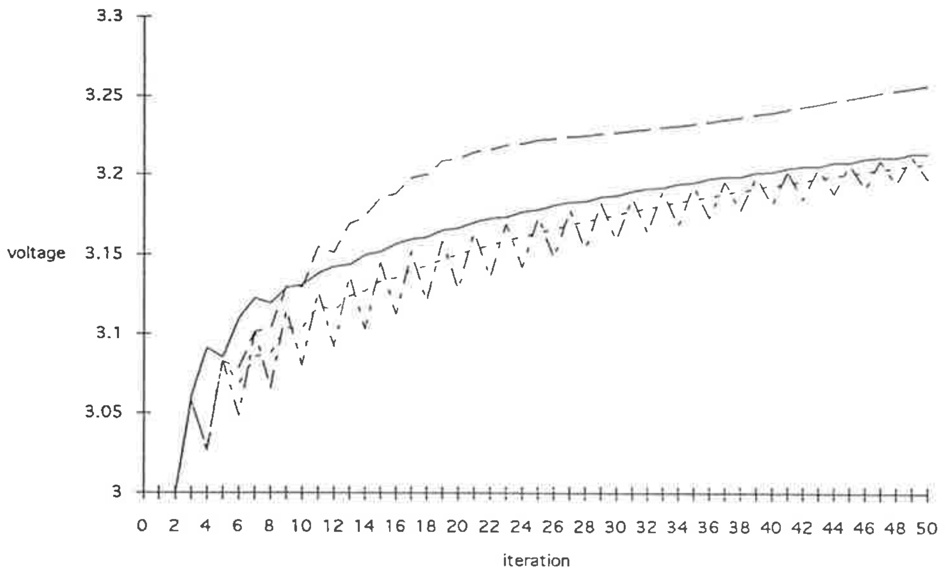


Figure 10.9: Startup Voltages on N face of element 77 Compared

Method	Iterations	RMS Dist.
pure bargain	662 iterations	1.14 mV
pure relax	265 iterations	130.5 mV
profit 1×5	59 iterations	51.0 mV

where the *profit* 1×5 method is described in the next chapter.

10.3.2 Relaxation Method

Normally the *relaxation method* is applied to the linearlized differential equatiations. Here in the *quasi-static approximation* there are no differential equations so it is applied directly to the array for purposes of comparison with the other methods.

Using the *relaxation method* the voltages show a characteristic step-like convergence; the voltages on each face of an element inexorably but slowly moving towards the final value. The maximum residual does not decrease as inexorably, however, since liquidating the residual at only one face can allow it to increase at another. This behaviour is shown in figure 10.10 on page 146 and figure 10.11 on page 146.

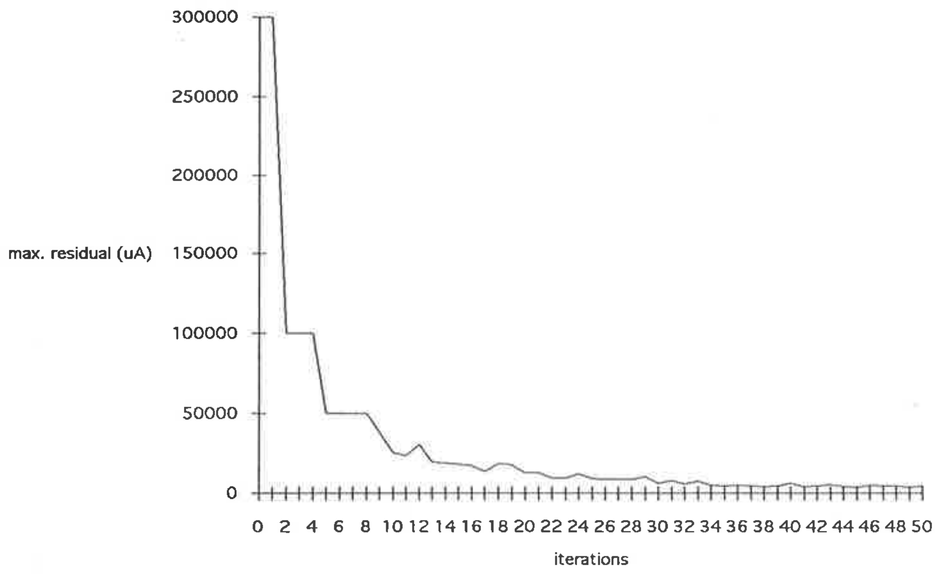


Figure 10.10: Pure Relaxation Method - Max. Residual Reduction

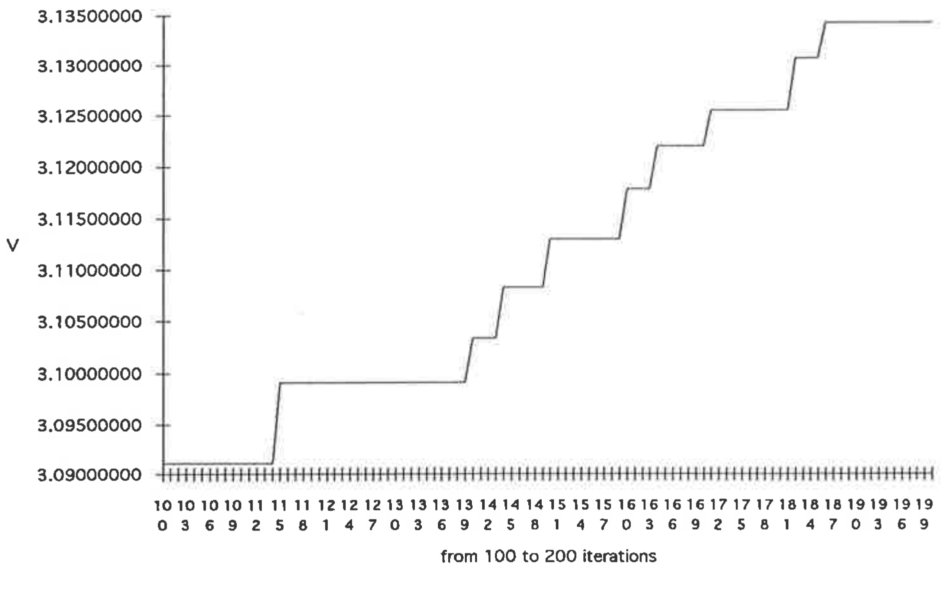


Figure 10.11: Pure Relaxation Method - Face Voltages as Iterations Increase

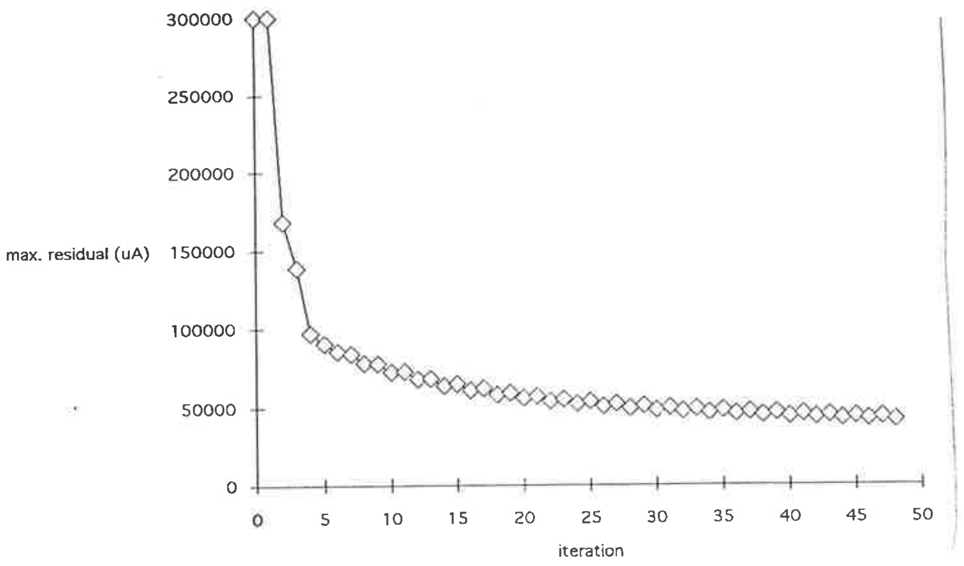


Figure 10.12: PURE BARGAIN METHOD - MAX. RESIDUAL REDUCTION

10.3.3 Pure Bargain Method

In the system using this plain *bargain method* the voltages show a characteristic zig-zag convergence; voltages on opposite faces of an element doing opposite adjustments, because the current equations for the current mismatches on all four faces are being solved *independently* of one another; but the maximum residual steadily decreases, eventually taking 1784 iterations to reach the 1μA convergence criterion. This behaviour is shown in figure 10.12 on page 147 and figure 10.13 on page 148.

To put the voltage behaviour in perspective over the whole waveform, figure 10.14 on page 148 shows the development of selected nodes over many iterations. However, this does not reveal the fine detail, which is presented in figure 10.15 on page 149. This voltage behaviour continues for a very long number of iterations in this fashion, no matter which nodes are examined.

In the mechanical case there are oscillations produced, too, so it is expected that here the voltages will show the same oscillations. The maximum residual also shows the oscillations, superimposed on the step-like reduction characteristic of the *relaxation method* and is shown in figure 10.16 on page 149.

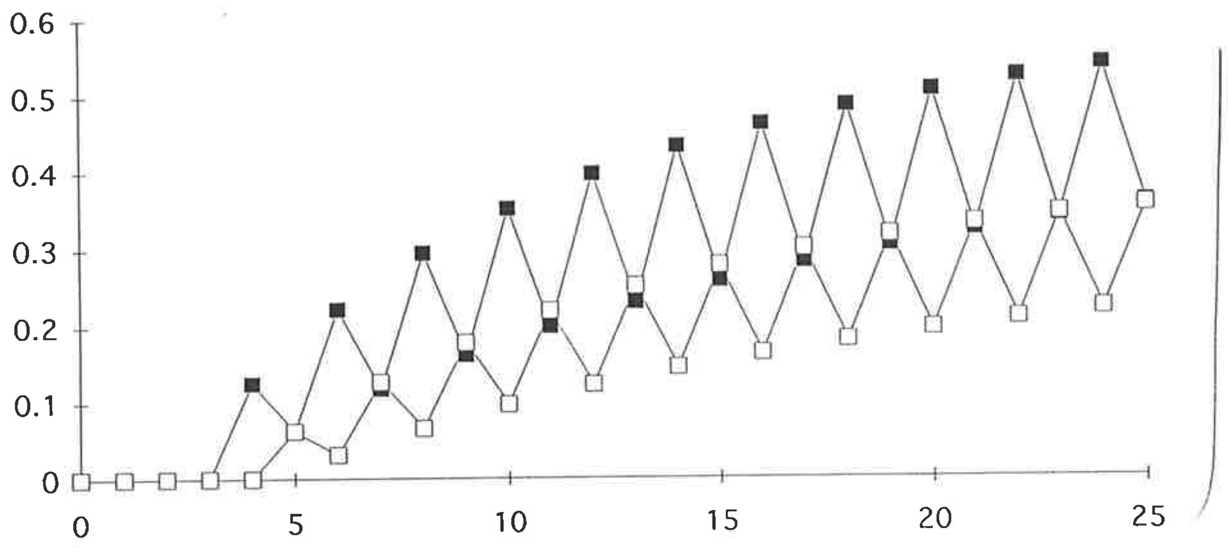


Figure 10.13: Pure Bargain Method - Startup Voltages on Opposite Faces

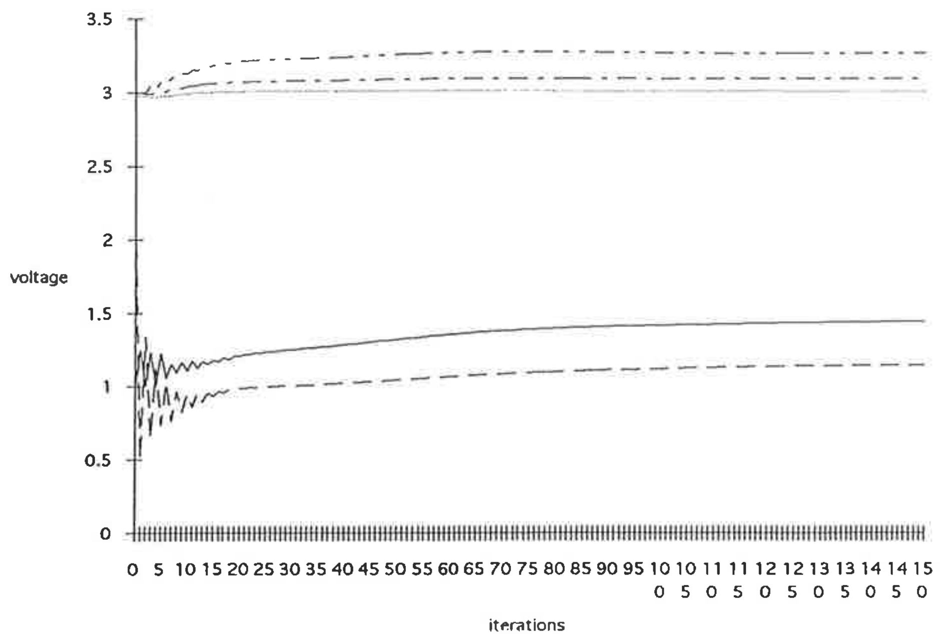


Figure 10.14: Bargain Method: Selected Voltages from Startup

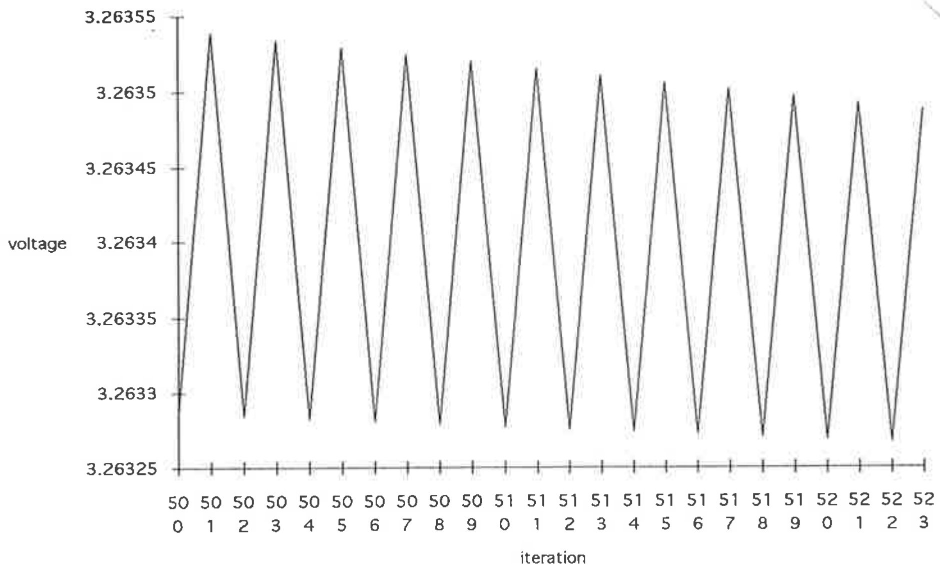


Figure 10.15: Pure Bargain Method - Voltage Convergence after 500 Iterations

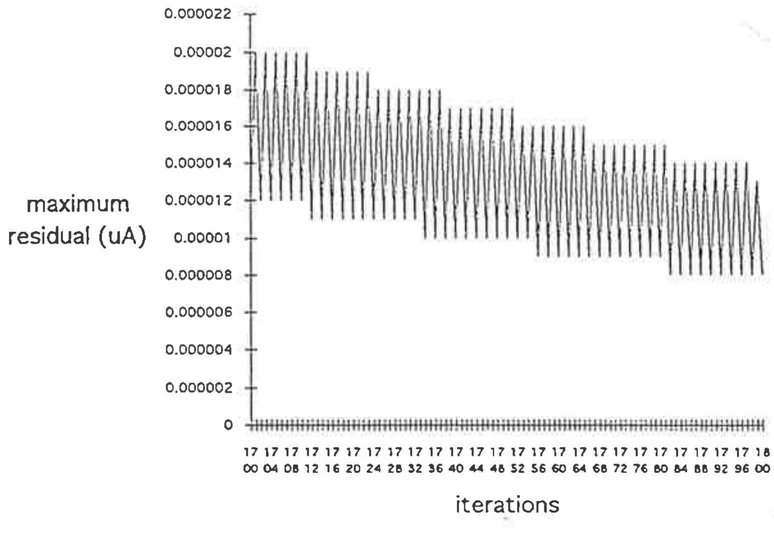


Figure 10.16: Pure Bargain Method - Maximum Residual after 1700 Iterations

The solution voltage comes down fast initially and then reduces more slowly. This is because the algorithm depends on the distance from the solution, it is driven by the current mismatch that it experiences - the smaller that gets, the slower the convergence.

10.4 Speedup Ideas

It must be admitted that none of this is particularly fast. So, for this array processor model study, how can speedup be achieved?

Over the whole driving voltage waveform, there are three main methods possible:

- speed up convergence between time steps;
- speed up convergence during one time step;
- develop new algorithms that work differently.

Each one of these is investigated in this thesis, and the results found are:

speed up convergence between steps requires getting close to startup point, which it will be shown cannot be done in general.

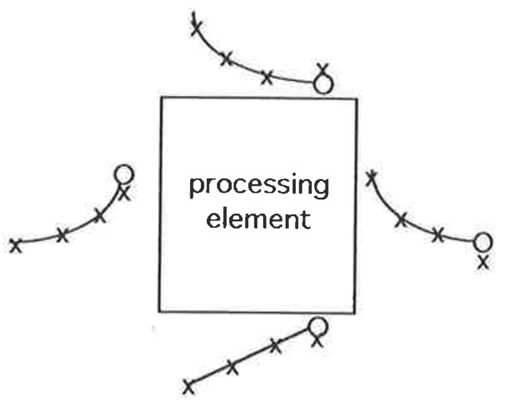
speed up convergence at one step leads to smoothing and lookahead.

develop new algorithms eventually produces the game theory approach that acts as a type of *simulated annealing* in that it jolts the system out of a rut to help it converge.

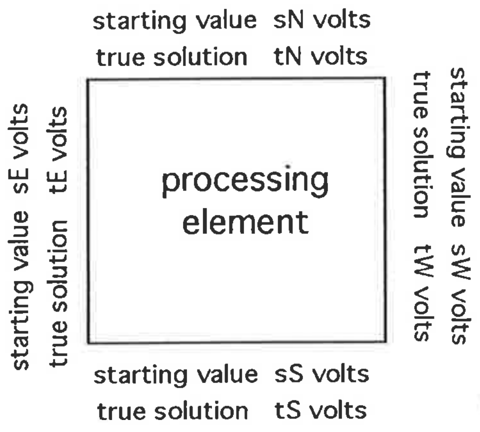
10.4.1 Speedup between Driving Steps

The aim is to closely predict the next set of voltages on each face for each element by using the last few values. This is found to fail because these predictions have to be actually very close indeed to the correct value, otherwise large increases in iterations occur.

The study that proves this is reported here. As illustrated in the diagram below, parabolic prediction of the next face voltage using the last 3 values



is attempted. A large number of random starting points is chosen for the face voltages throughout the array. As illustrated next, the sets of starting



voltages on the faces of each element - s_N, s_E, s_S, s_W - are chosen at random. Then the system is run until 1μA convergence is reached. The distance of the initial set of startup voltages from the true final solution is measured by the Root Mean Square voltage difference per face per element

$$\sqrt{\sum_{elements} ((s_N - t_N)^2 + (s_E - t_E)^2 + (s_S - t_S)^2 + (s_W - t_W)^2)/4/64}$$

Each point on these graphs represents a sampled starting set of node voltages and the corresponding number of iterations needed to get to within 1μA convergence. In this way, the effect of starting some distance from the true solution on the final number of iterations can be guaged.

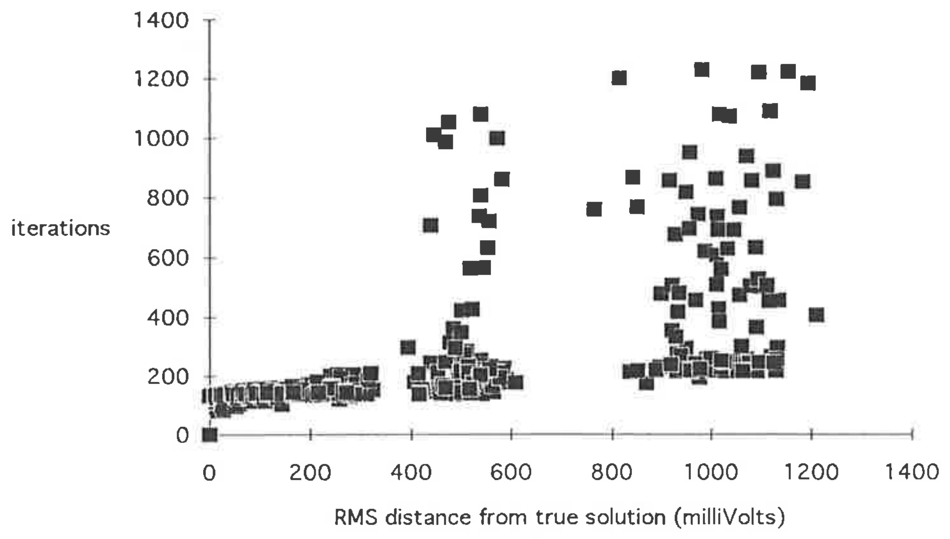


Figure 10.17: How Iterations Increase as Startup gets away from True Solution

The results show a very tight spread over a considerable range of startup position, shown in figure 10.17 on page 152. This shows that it is necessary to start extremely close to the actual solution to get a significantly reduced number of iterations - if just a small RMS distance away, which is quite likely, the number of iterations increases very sharply, and so the startup point might just as well be a long way from the actual solution, without taking any trouble to get it close. This is seen more clearly in the expanded section near the start of the plot, in figure 10.18 on page 153, where the iterations jump from zero to almost 140 within a quarter of a millivolt of the true solution.

Also shown, in figure 10.19 on page 153, is the result over a wide range of RMS distances from the true solution, showing that there is a point after which the iterations, and the standard deviation of the iterations, start to increase at a further rate.

It is concluded that this inter-step prediction is very unrewarding since it is found to be very likely to get too far from the true value by parabolic interpolation. The results from 1000μA convergence criterion, done to check

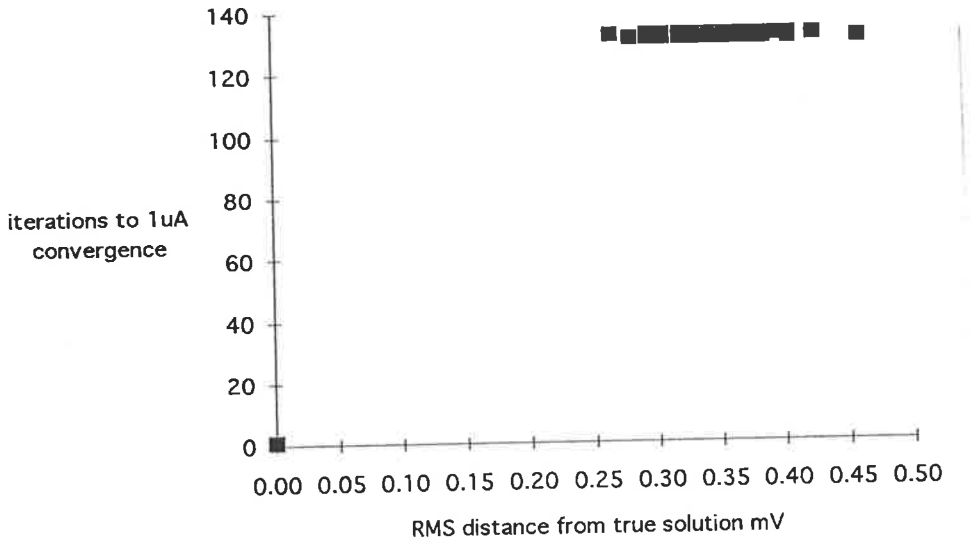


Figure 10.18: Detail of Iteration Increase close to True Solution

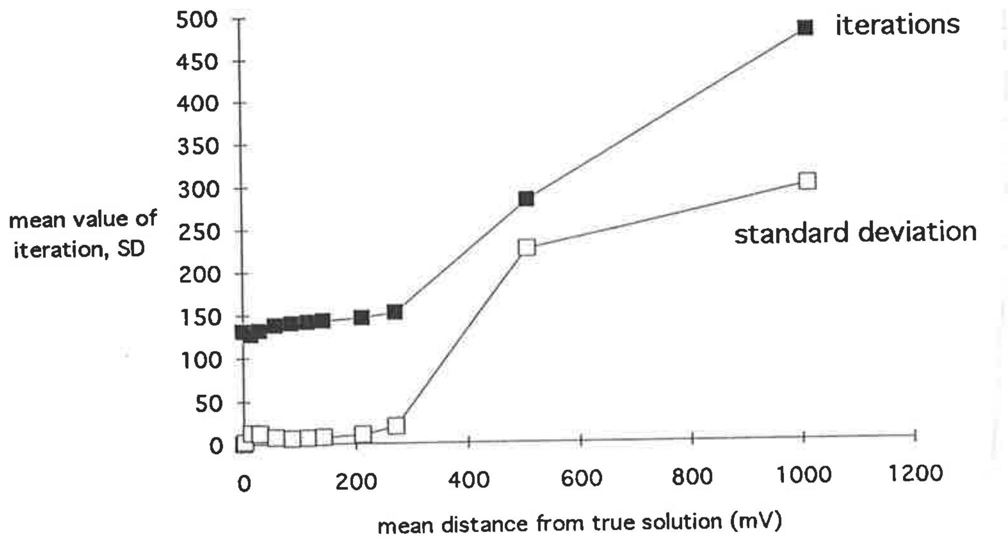


Figure 10.19: Statistics of Iterations as a function of Distance from True Solution

consistency, are slacker but show a similar shape - the startup has to be very close to benefit from reduced iterations and this is found to be impractical.

10.4.2 Speedup by Smoothing Method

Given the *Bargain Method*, there are two ways of possible improvement: one is to consider smoothing the inevitable oscillations over past values; the other is to consider smoothing somehow over present values.

Past values leads to the smoothing+lookahead method in this chapter.

Present values leads to the game theory algorithm in the next chapter.

The oscillations continue on a small scale for a very long time, just reducing slowly, as already seen in figure 10.16 on page 149.

Since it is impractical to start off very close (for all faces) then other methods must be tried. In an attempt to dampen the inevitable oscillations arising from independent and uncoordinated solutions on each face, it is possible to examine:

- using a linear predictor;
- weighting of previous values;
- least squares best fit.

10.4.3 Linear Predictor

This uses a criterion like adaptive linear predicting in signal processing [Pap91]. The new voltage on each face is predicted from the voltages obtained at the last 3 iterations by using coefficients a, b, c in the form

$$V_{predicted} = aV_{-1} + bV_{-2} + cV_{-3}$$

so that when the next values are smoothed and the actual value computed, a, b, c are chosen so that the mismatch is minimised by the differential process

$$\delta \sum_{faces} (V_{predicted} - V_{actual})^2 = 0$$

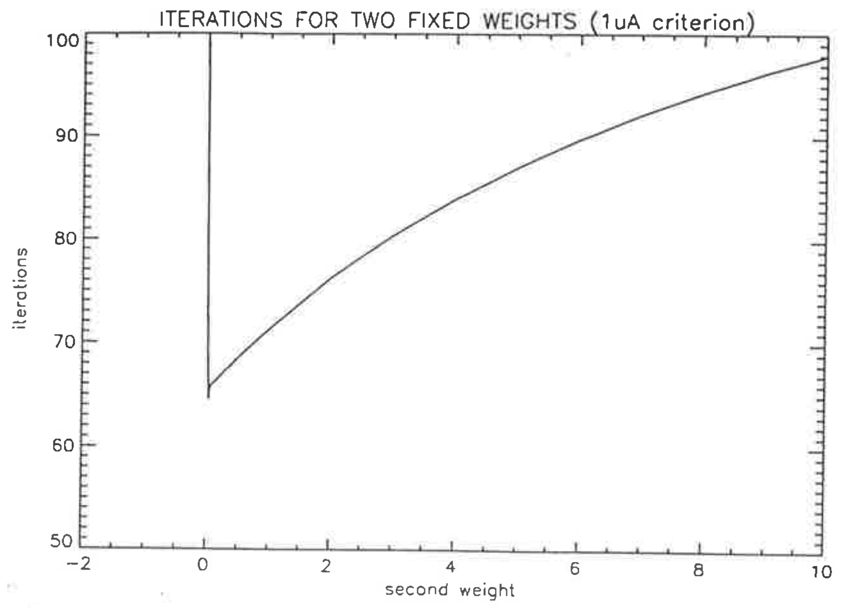


Figure 10.20: Effect of Weighting Previous Value of Voltage

with respect to a, b, c. This is solved for a, b, c and the new voltage predicted. It is found that this scheme produces a very erratic set of predictions, and ultimately takes more, not less, iterations. So the linear predictor idea fails, presumably due to the erratic and zigzag nature of the voltages as they converge.

10.4.4 Weighting of Previous Values

The next attempt to dampen the oscillations is by weighting the next guess by various previous values, for example the previous 1 or 2 values.

The results in figure 10.20 on page 155 show that weighting the current face voltage by the previous one *via* the weight w thus:

$$V_{next} = V_{current} + wV_{previous}$$

produces a very strange graph indeed - in fact, only if the weight $w = 0.042$ is a sharp minimum achieved.

This sharp minimum, occurring at very low weighting, is confirmed by the following results, where this time 2 previous values are used i.e.

$$V_{next} = V_{current} + AV_{-1} + BV_{-2}$$

where various combinations of weighting of 2 previous values are tried.

The results are seen to be similar at two different convergence criteria and show minima for small values of the previous valued point only. The various graphs shown in figure 10.21 on page 157 represent different values of the coefficients A,B where: A goes from 0 to 10 along the horizontal axis; B goes from 0 to 10 for each curve, with $B = 0$ at the bottom and $B = 10$ at the top. The iterations are given as a percentage of the no-weight case for ease of comparison.

10.4.5 Least Squares Best Fit

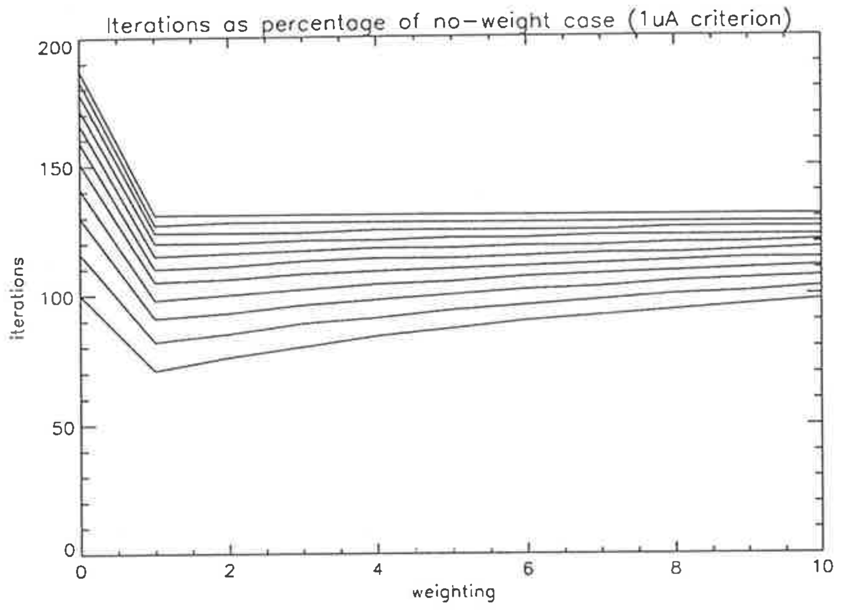
It is seen that it is not possible practically to find the optimum weight by the methods of linear prediction since the curve is not differentiable at the optimum. What can be tried next? Since the voltage oscillates between two series of values, then the simplest averaging method is to fit a straight line of best fit through the last few points. This has been tried, and it is found with the last 6 points a fairly stable result is obtained. Thus on all 4 faces of each element the next voltage to start with is got from fitting a least-squares best fit through the voltages at the last 6 iterations.

The top figure 10.22 on page 158 shows how fast the array converges when there is no LSQ smoothing. Contrasted with this is the marked improvement using the LSQ smoothing technique, shown in the bottom graph for the same case.

The next graph in figure 10.23 on page 159 shows the effect of this 6-point LSQ smoothing on a typical startup face voltage - the LSQ takes place from iteration 6 onwards, and the zig-zag behaviour observed before in figure 10.13 on page 148 has been substantially eliminated.

As the iterations proceed the voltage on the N face of element 72 is shown in figure 10.24 on page 159 regularly and smoothly converging.

Interestingly the overall pattern of convergence expressed in terms of reduction of maximum residual has a strong linear-log character (also observed



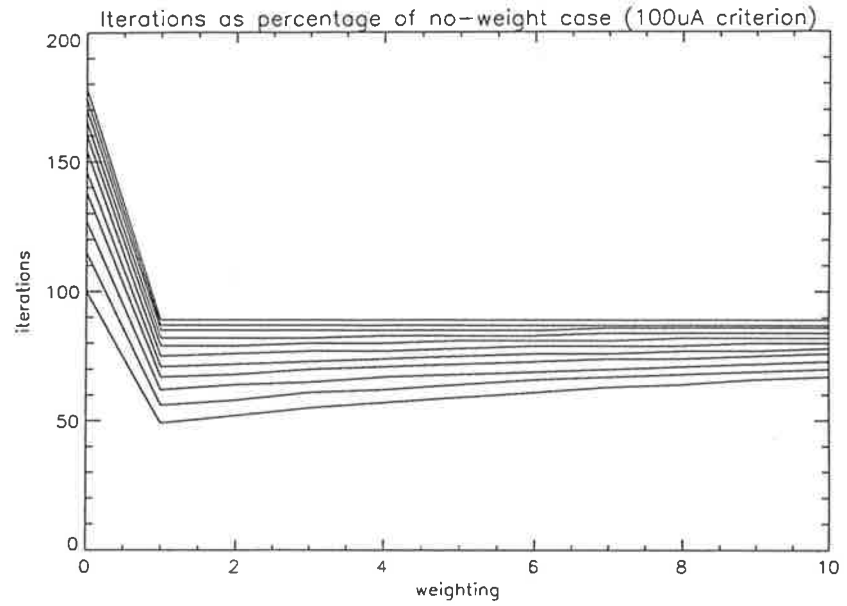
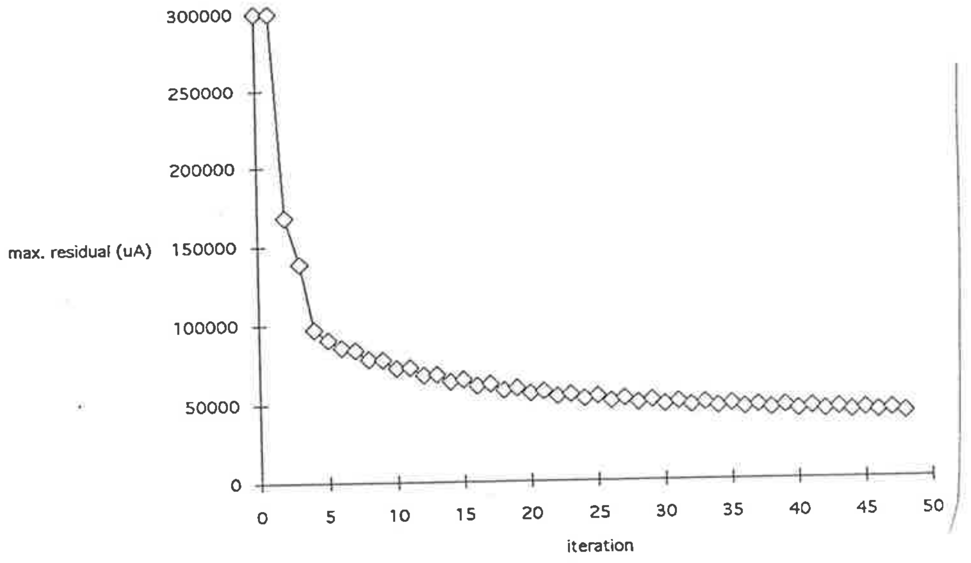


Figure 10.21: EFFECT OF WEIGHTING TWO PREVIOUS VALUES - 1μA CRITERION (TOP) AND 100μA CRITERION (BOTTOM)



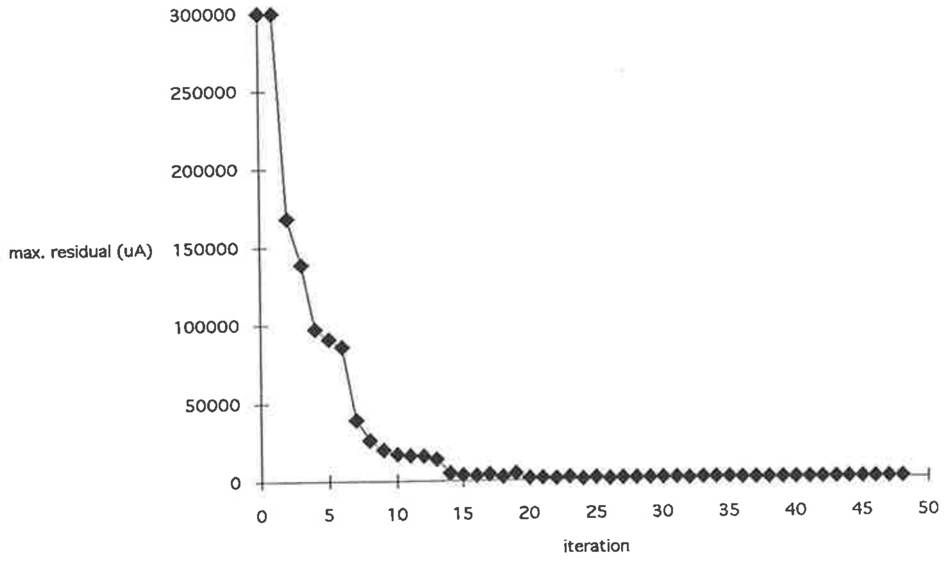


Figure 10.22: Effect of Applying 6-Point Least Squares Smoothing to Bargain Method: No Smoothing (top) and Smoothing (bottom)

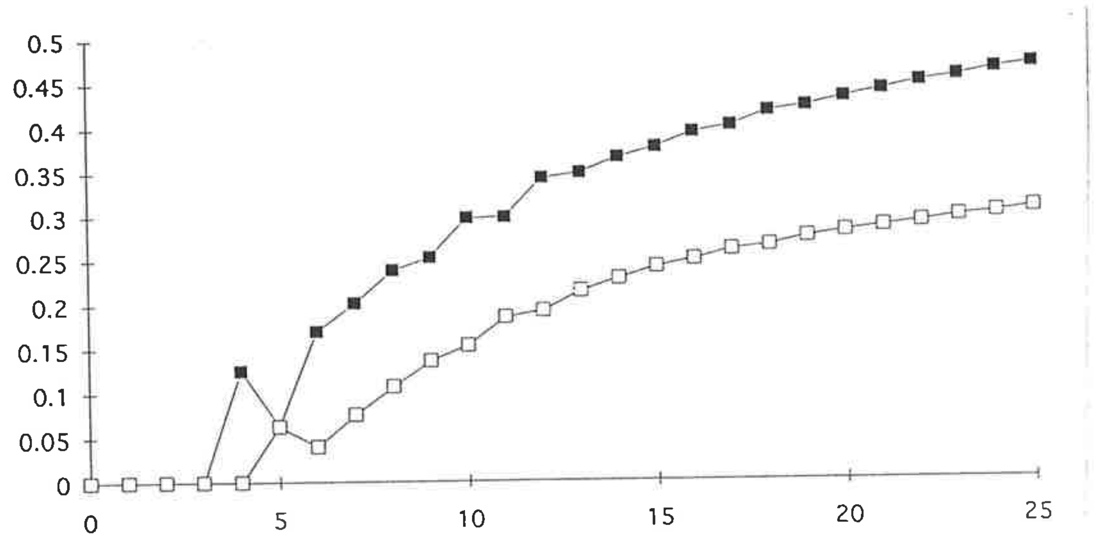


Figure 10.23: EFFECT OF APPLYING 6-POINT LEAST SQUARES SMOOTHING TO OPPOSITE FACE VOLTAGES

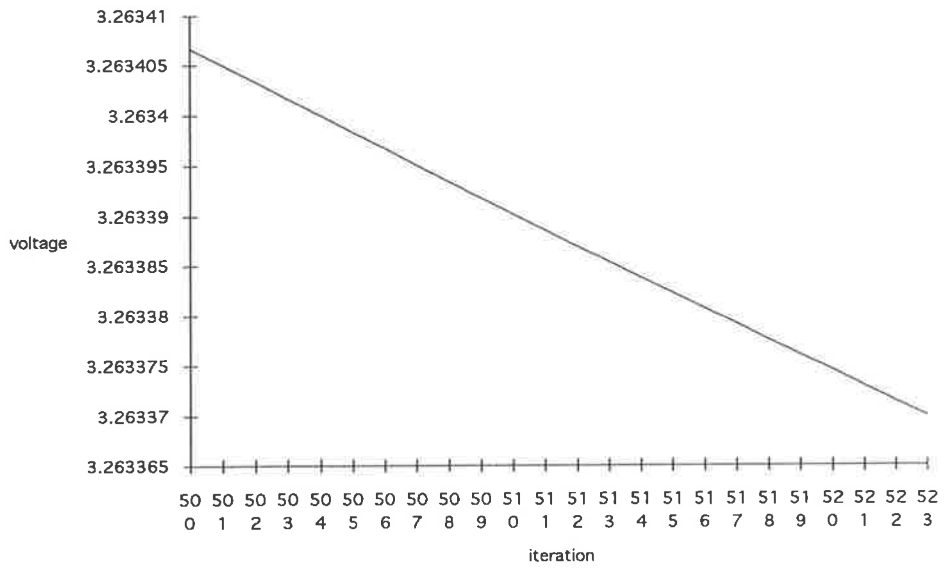


Figure 10.24: VOLTAGE IN BARGAIN METHOD WITH SMOOTHING

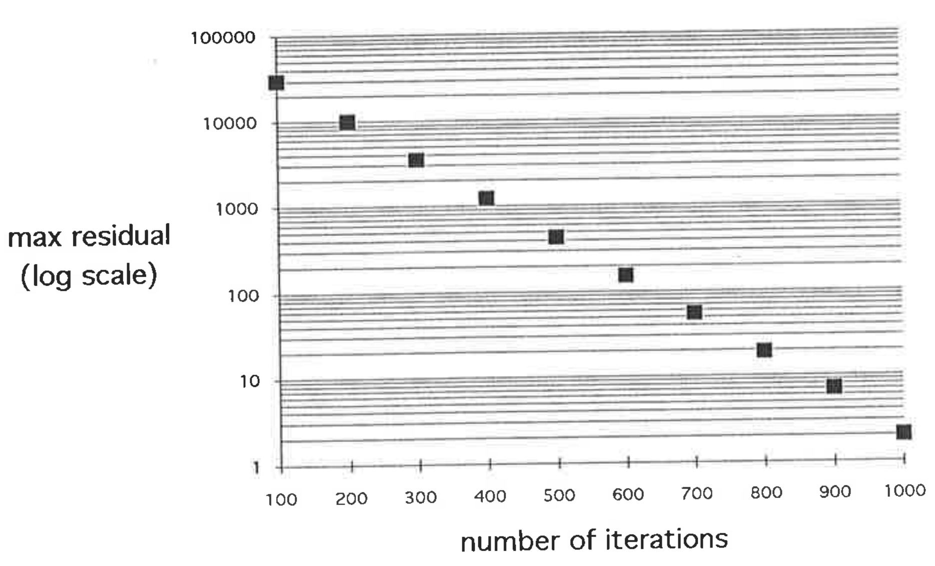


Figure 10.25: BARGAIN METHOD - LINEAR LOG REDUCTION OF RESIDUAL

in other similar systems) as shown in figure 10.25 on page 160.

10.5 Look-Ahead Ideas

Since it has been seen that a trend is established along the two tangents to the upper and lower set of oscillating points, then it is natural to expect that extrapolation of the predicted value could get to the correct value more quickly. Ideally, a different amount of extrapolation would be needed for each element and each face, *i.e.* an adaptive scheme is required; however, to try out the extrapolation idea, just the same value for all elements is used, on the basis that if it will not work for this case and reduce the iterations, then investigation of further adaptive cases will not be fruitful; whereas if it does work then there is scope for future research into adaptive methods.

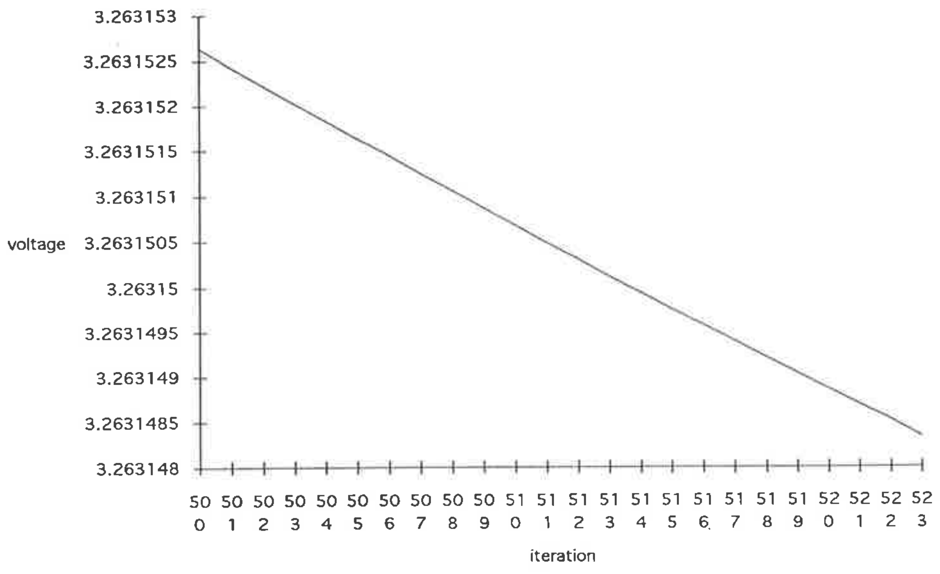
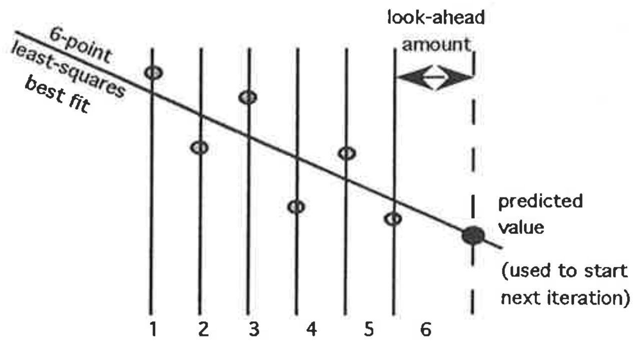


Figure 10.26: VOLTAGE SMOOTHED WITH LOOK-AHEAD APPLIED

The lookahead scheme works as shown in this diagram.



6-point least-squares best fit
look-ahead amount
predicted value
(used to start next iteration)
1
2
3
4
5
6

The results are unexpected. It is found that in fact it is not possible to anticipate too far ahead. If extrapolation ahead by more than 1 iteration is attempted then an increase in the number of iterations occurs again instead of a decrease. Improvement in the number of iterations can be quite dramatic, however, depending on how much lookahead is done and when.

The effect on the voltages is shown in figure 10.26 on page 161.

The effect on the convergence, measured by the maximum residual, is shown in figure 10.27 on page 162.

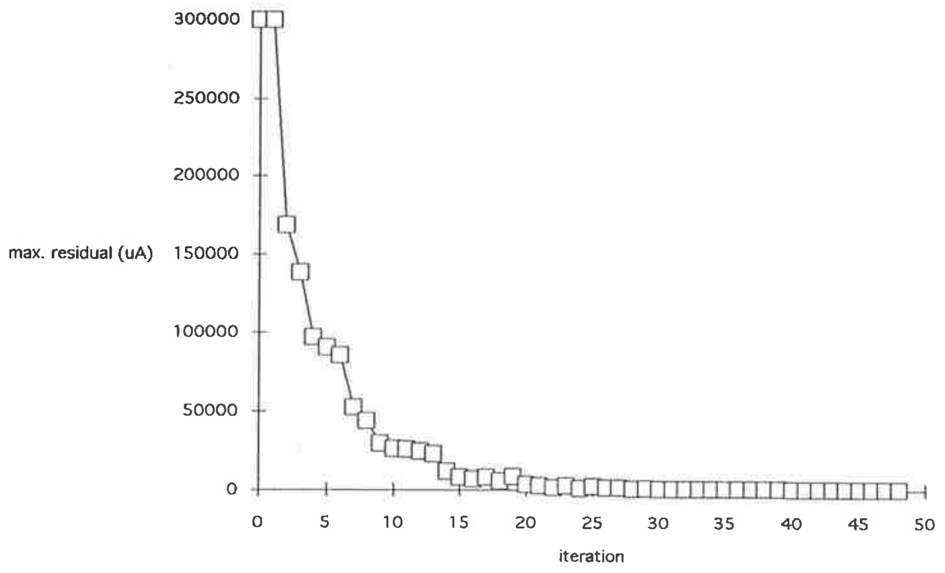
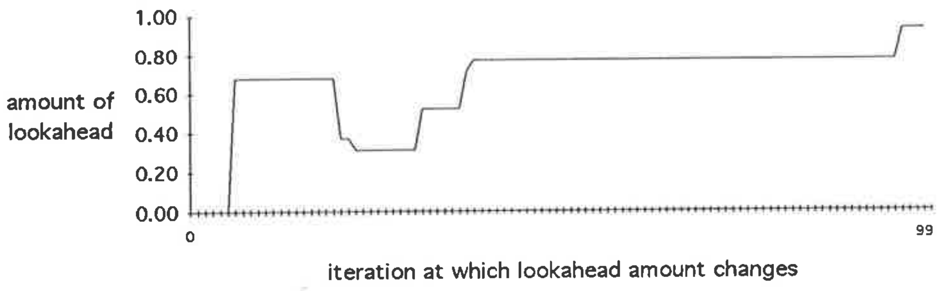
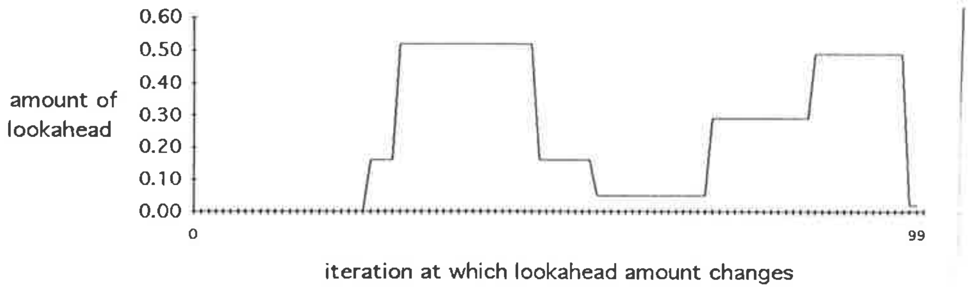


Figure 10.27: RESIDUAL SMOOTHED WITH LOOK-AHEAD APPLIED

A certain amount of experimental investigation found that improvement in iterations depended on how much lookahead is used and where in the iteration process it is used. It is found, for example, that this figure below is



the profile producing minimum iterations (106 iterations with 7 changes) and this figure next is the profile producing maximum iterations (581 iterations



with 7 changes). Note that the minimum curve zigs where the maximum curve zags, as would be expected if they are producing opposite effects.

Thus a systematic study of lookahead tactics was carried out. In default of the ideal situation[4] an exhaustive random experiment chose both: a random set of LA[5] values between 0 and 1 (since it is already known that > 1 is no good); and a random set of iteration points at which to change the LA value.

500 samples were run to convergence at the 10μA criterion, the results being summarized in figure 10.28 on page 164.

The standard deviation of the spread also is shown in figure 10.29 on page 164.

Both these results show that:

- if a LA tactic is to be done, then it is best not to do it timidly, *i.e.* do lots of LA changes;
- it is possible do badly by random methods;
- it is possible to do much better than the average random choice;
- thus research into an optimum adaptive algorithm is highly desirable.

[4]an adaptive method for each element and for each face

[5]Look-Ahead

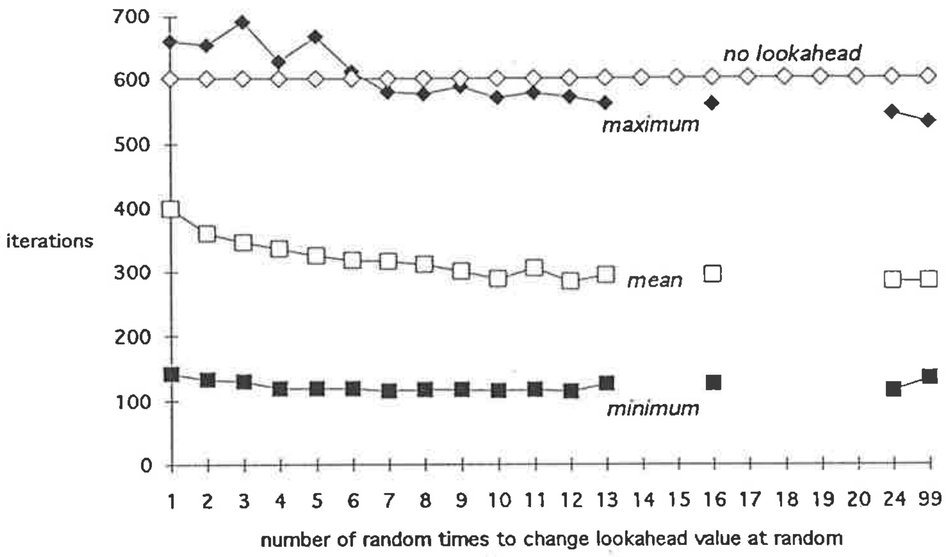


Figure 10.28: SUMMARY OF LOOKAHEAD STUDY RESULTS

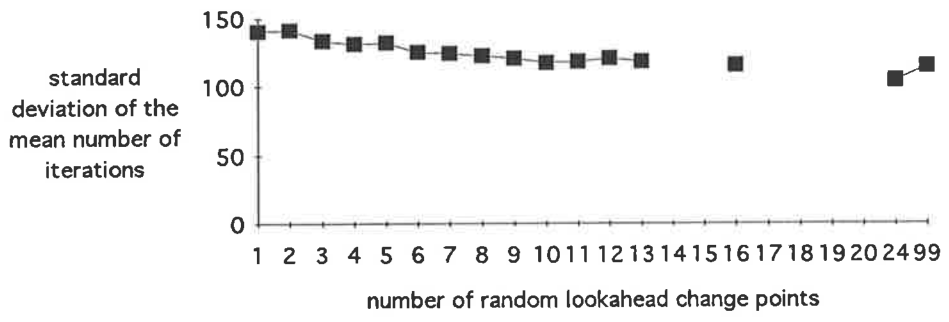


Figure 10.29: VARIABILITY OF RANDOM LOOKAHEAD ITERATIONS

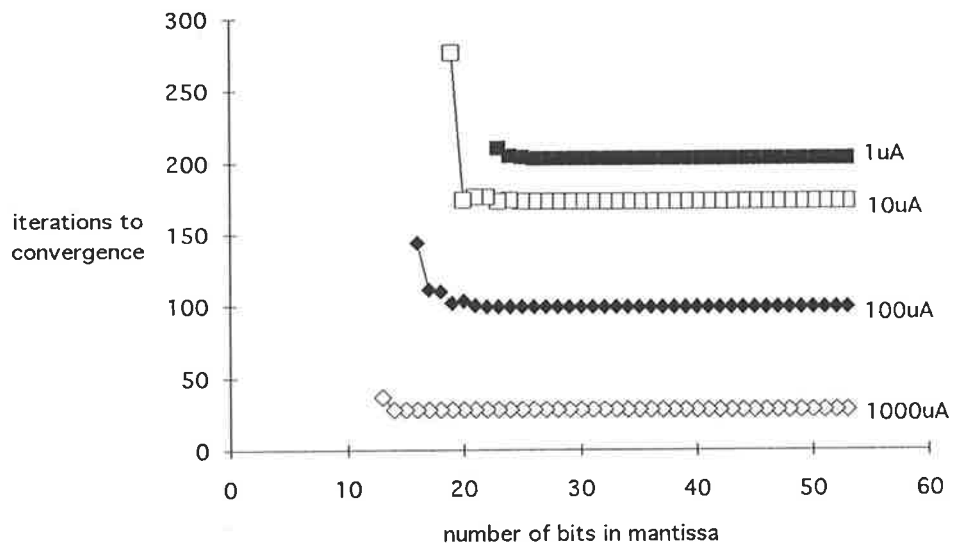


Figure 10.30: BARGAIN+SMOOTH+LOOKAHEAD: CONVERGENCE AS FUNCTION OF NUMBER OF MANTISSA BITS

10.6 Accuracy *versus* Number of Bits

Figure 10.30 on page 165 shows how the *bargain + smoothing + lookahead* system convergence behaves as the number of mantissa bits is reduced; this information is needed to decide the architecture of processing elements.

A straight line can be drawn through the regions where convergence suddenly fails to occur at each criterion; this the prescribes the minimum number of mantissa bits necessary for that criterion to be workable. The critical number of bits thus found is shown in figure 10.31 on page 166.

10.7 Operations Count per Iteration

Since all the operations with real numbers are masked in the C program to obtain the effect of using various mantissa bits, it is a simple matter to *count* all the operations at the same time. This gives a basis for estimating how long the real array simulator, if built, would take under various conditions. This data is used in Part 4 to compare the various strategies that are investigated

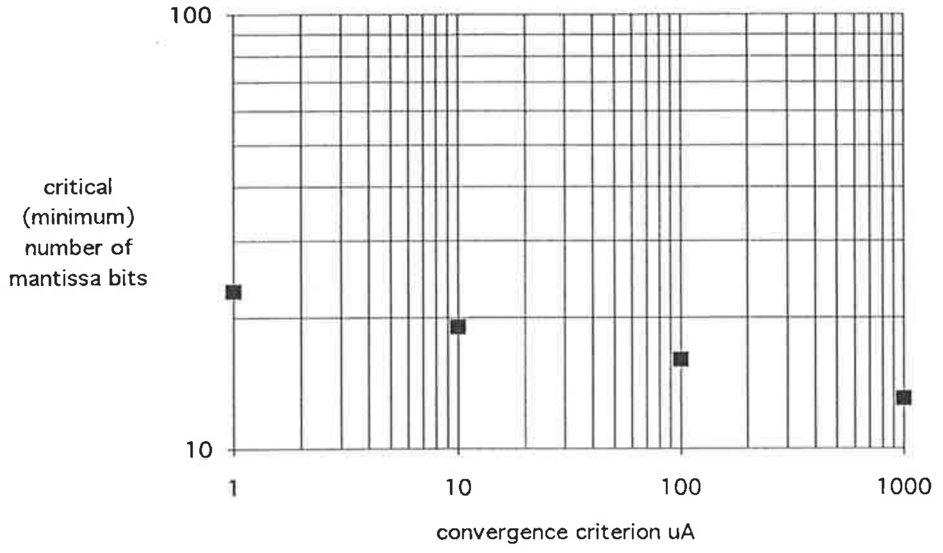


Figure 10.31: Number of Bits Needed in Processing Element

in Part 3.

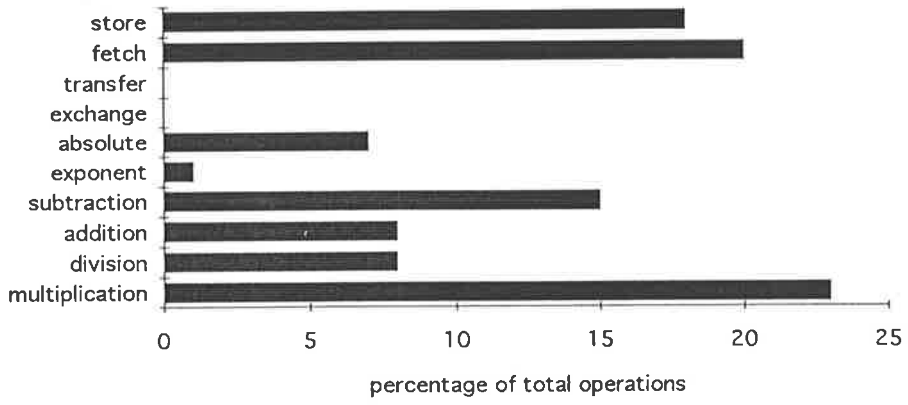
The *bargain+smooth+lookahead* method, across a wide range of mantissa bits[6] is found to use a steady number of operations per iteration, summarized in this table.

Operation	Number per Iteration
multiplication	500
division	312
addition	280
subtraction	218

The next bar graph summarises the relative percentage of operations, for 53 bits over the whole simulated waveform, on average 100 iterations per

[6]from 12 bits up to the maximum of 53 bits

timestep.



To give a rough idea of time taken, the total number of counted operations is around 400 million operations per waveform for the 200μA criterion, and around 270 million for the less stringent 400μA criterion; once an architecture for the processing elements is known, the real simulator time can be estimated from the cycle time and the cycles per operation.

For interest, rough estimates for the *bargain+smooth+lookahead* method are given in the next table, assuming a 100 nanoSecond clock cycle and approximately 200 multiply operation cycles per bit.

BITS	CRITERION	TIME	MSEC	ITERATIONS
16	500	0.7	18	9
16	400	0.8	18	10
16	300	0.9	18	12
16	200	1.1	18	15
24	500	1.0		9
24	400	1.1		10
24	300	1.2		
24	200	1.5		
28	500	1.2	33	9
28	400	1.3	33	10
28	300	1.5	33	11
28	200	1.8	33	13
28	100	2.5	33	18

In this table, BITS is the number of bits in the mantissa; CRITERION is the convergence criterion in μA; TIME is the estimated mins. to carry out 100 simulation; MSEC is the real time per iteration in milliSeconds; and ITERATIONS is the mean number of iterations per timestep over 150 time steps.

In addition, rough estimates for: the *pure bargain* method, 53 bits, 5μA convergence is 49 mins., with an average of 170 iterations over 150 timesteps; the *bargain+smooth+optimal lookahead* method, 53 bits, 5μA convergence is 16 mins., with an average of 170 iterations over 150 timesteps; and the *bargain+smooth+optimal lookahead* method, 16 mantissa bits, 5μA convergence is 40 seconds, with an average of 9 iterations over 150 timesteps, taking 18mSec per iteration.

The convergence accuracy, as measured by the RMS[7] distance over the whole waveform from the true values of Q and $\bar{Q}$ at various convergence criteria for 16 bits is indicated in the table:

Convergence Criterion	Q Accuracy	Q Accuracy
500 μA	0.269 mV	0.124 mV
400 μA	0.205 mV	0.103 mV
300 μA	0.162 mV	0.083 mV
200 μA	0.119 mV	0.064 mV

10.8 Conclusions for Bargain and Smooth and Lookahead

The methods investigated converge in a characteristic zig-zag fashion. Most iterations are needed when the driving voltages are most out of balance. Various tactics were tried to improve convergence rates in these areas, but it was found that simple least-squares smoothing was remarkably effective in the first instance.

Success with other methods was produced on an *ad hoc* experimental basis. However, general success with those methods depends on the application of adaptive prediction techniques to get the best use of each technique at each iteration. The *lookahead* tactic particularly can produce large improvements under these conditions.

The behaviour of the best algorithms was limited by the number of mantissa bits used: any number from 12 to 24 bits were necessary depending on

[7]root mean square

the accuracy of convergence required.

It was found that the bargain method and its variants have a characteristic operation-count profile, which can be used to estimate the total real time for 100 simulations. Thus it can be used to compare the tactics, and to compare them with other algorithms. It can be very useful in assessing algorithmic improvements, since fewer iterations achieved can be more than offset by the extra time taken by calculations in a more complex algorithm.

There are storeage implications, speed and cost trade-offs involved in the various tactics examined. For example, varying weights meant that using the same weight for all faces of all elements requires storage of 3 previous voltages - 3 real numbers per element; whereas a more flexible tactic of using a different weight for each face of each element, while no doubt producing a faster convergence, would require storage of 3 weights per face - 12 real numbers per element - thus making the element more complex and expensive.

Concentrating on interstep speedup is not found to be worthwhile because of the need, and the inability, to get very close to actual values of all nodes to reduce iterations much. Therefore effort needs to be concentrated on speeding up the internal iterations within one timestep instead.

The improvement is greatest only in the early stages of the iteration. Later, as it gets closer it gets slower, as usual. So it appears that the convergence might be considered in two fairly distinct phases, the early one being quite substantially sped up, the latter requiring a different approach.

Chapter 11

Game Theory

In this chapter I study the application of economic game theory to the problem of array convergence and find that it works surprisingly well.

11.1 Why try Game Theory?

The germ of the idea that *Game Theory* might be applicable to speeding up convergence, or even to the problem of just achieving convergence, came from the fundamental reinterpretation of the Newton-Raphson iteration technique, described in Appendix C. It seems that there are still things to be discovered about this technique, as François Robert confirms in his exploratory work on discrete iterations [Rob87], writing:

> *"One may for example refer to the algorithmic . . aspects of Newton's method . . . seems to me to be the most fascinating algorithm in numerical analysis . . . shows a remarkable practical efficiency even though there exists relatively few global convergence results."*

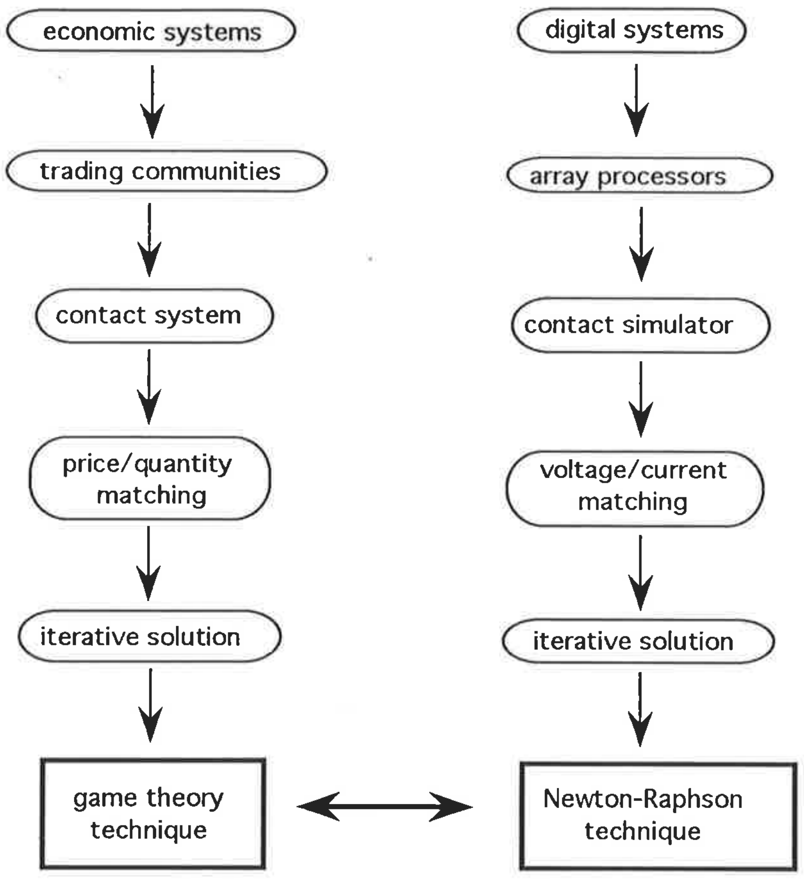


Figure 11.1: Comparison of Modelling Methodologies

Consider the possible method of modelling the economic behaviour of communities and the analogous possible method of modelling arrays of processing elements in digital microelectronics suggested in figure 11.1 on page 171.

It has been discovered in this thesis that there is a strong similarity between iterative economic-bargaining and the Newton-Raphson iteration technique (shown by the bold arrow at the bottom of the figure). The similar structure of each modelling chain above the bottom boxes suggests that there might exist a similar strong correspondence at every level of the chain.

Thus it is possible that a more formal application of game theory methods might prove useful, by adapting known bargaining strategies to the purpose of reducing the oscillations that occur during some of the iterations.

The model used for this study is the flipflop circuit at the start-up situation because it takes by far the most iterations to converge and in which the most speed improvement is possible.

11.2 Elements of Game Theory

According to Moeschlin and Pallascke [MP80] and Ichiishi [Ich83] a *Game* has the features defined and explained below.

11.2.1 Definition of a Game

players: there is a set of more than one decision-makers, called *players* ;

moves: at specified instances, one or more players must make decisions by choosing amongst a set of specified alternatives;

choices: each situation determines which of the players is to move, (the moves may be simultaneous) and the range of choices;

endplay: certain specified situations define the end of that particular play of the game;

payoffs: the outcome of each play has payoffs;

strategy: rational players strive to maximise their expected payoff.

Furthermore, game theory recognises that if a choice must be made between two actions, then the realm of decision-making is that of:

certainty if each action is known to lead invariably to a specific outcome;

risk if each action leads to one of a set of possible specific outcomes, each outcome occurring with a known probability;

uncertainty if either action or both has as its consequence a set of possible specific outcomes, but where the probabilities of these outcomes are completely unknown or are not even meaningful.

11.3 Development of Game Theory Model

The above definitions suggest that each processing element is an individual making decisions under uncertainty, confirming that useful convergence strategies might be sought in the literature of game theory.

It has been shown how the NR[1] iteration can be re-interpreted in GT[2] terms as a bargaining strategy. If economic game theory methods are to be tried then it is necessary to identify quantities from the electronic sphere that are analogous to quantities in the economic sphere. This identification can only be tentative, like the application of this whole technique, and its justification depends on the final results of the simulations.

11.3.1 Identification of Game Theory Quantities

It seems reasonable to try the correspondences below, which are summarized in figure 11.2 on page 174:

price $\Leftrightarrow$ voltage The voltage between adjacent faces, being initially in dispute between the processing elements and having to be determined by an iterative process until equal, suggests that the voltage in electronics is the analog of price in economics.

quantity $\Leftrightarrow$ current Since at the negotiated price the currents flow that are equal from one element to the other, it is reasonable to suppose that the current in electronics is the analog of quantity in economics.

outlay $\Leftrightarrow$ power Since total payment out of one face of an element is in economics just $price * quantity$ then it is reasonable to take it that the outlay is the analog of $voltage * current$ or *power* in electronics.

exchange rate $\Leftrightarrow$ mathematical weight In order to allow an emphasis to be placed on the inputs of some elements rather than others it is

[1] Newton-Raphson

[2] Game Theory

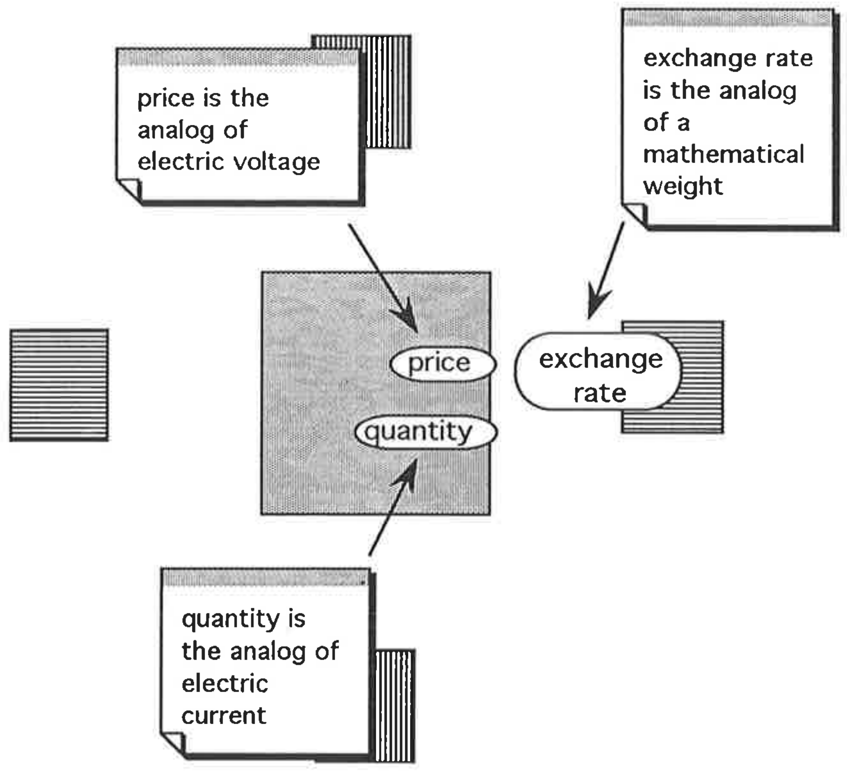

price is the analog of electric voltage
exchange rate is the analog of a mathematical weight
price
quantity
exchange rate
quantity is the analog of electric current

Figure 11.2: TENTATIVE IDENTIFICATION OF ANALOGUES

useful to establish what is the analog of an exchange rate between countries, or a risk loading between individuals.

profit $\Leftrightarrow$ power The nett outflow, or the algebraic sum of the outlays through each of the four faces, in economic terms leaves the element with a surplus or deficit, and since in electronics that same quantity is the nett power, or heat dissipated, in the element, then it seems natural initially to identify the maximisation of net profit as the criterion for elements. This suggests the adoption of the analogy *nett profit $\Leftrightarrow$ nett outlay.* This will be important when players attempt to maximise their payoffs.

11.3.2 Application to VLSI Simulation

So far the elements of the model have been tentatively identified with their economic counterparts. The next step is to identify each feature required of a *game* in terms of the SIMD contact model already investigated.

With reference to the features of a *game* described above, the following identifications with the processing array seem plausible:

players: The decision-makers *(players)* are the processing elements. There is a set of five decision-makers consisting of the main processing element and its four nearest neighbours, shown in figure 11.3 on page 176. Technically, this means that each element is participating in a *5-person non-zero-sum co-operative game.* Game theory literature shows there are no general theoretical results for such an advanced game, so everything that follows will be quite heuristic.

moves: Since the array is SIMD then all elements move simultaneously and independently within these alternative alterations.

choices: At each iteration, all elements make decisions by choosing amongst a set of specified alternatives. So far the only choice has been the result of the first NR iteration, but this is not realistic; there are at least 3

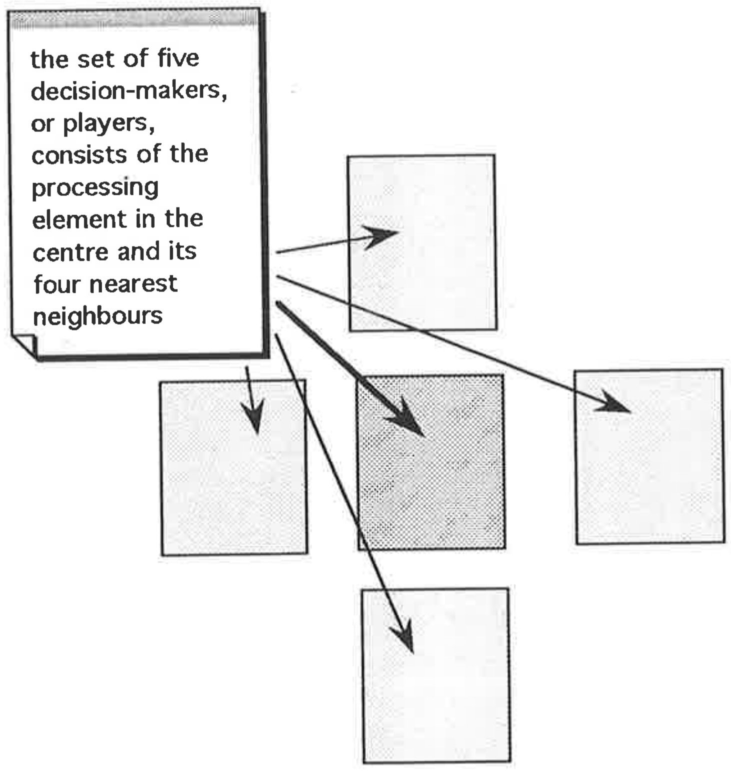

the set of five decision-makers, or players, consists of the processing element in the centre and its four nearest neighbours

Figure 11.3: Game Theory: Conceptual Arrangement

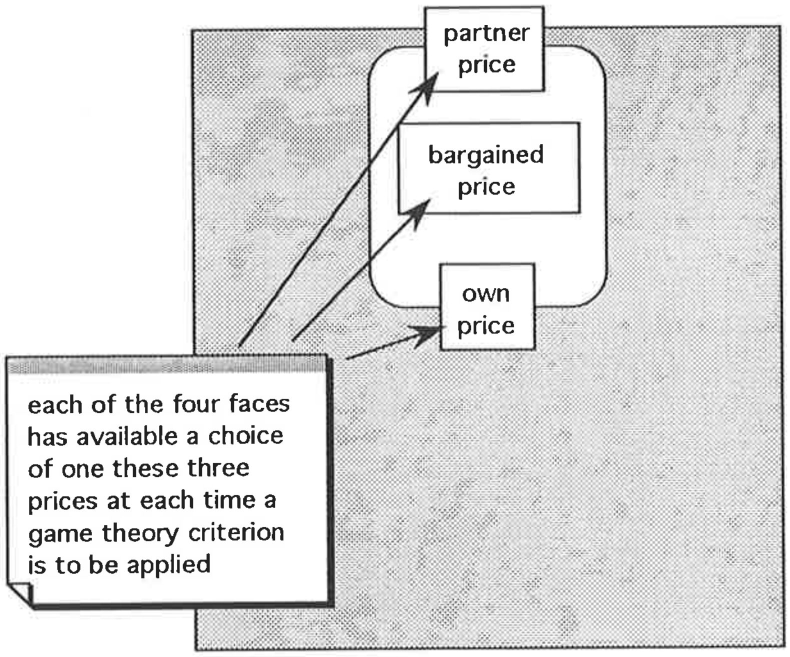

partner price
bargained price
own price
each of the four faces has available a choice of one these three prices at each time a game theory criterion is to be applied

Figure 11.4: OPTIONS AVAILABLE TO EACH ELEMENT

distinct choices each face may make at each iteration. Each element on each face can either choose **N-R** [3], **IP** [4] or **F-F** [5], as illustrated in figure 11.4 on page 177.

For the 4 independent faces of each element this yields $3 \times 3 \times 3 \times 3 = 81$ possible alternative combinations, given in this table:

combination	north	east	west	south
1	N-R	N-R	N-R	N-R
2	N-R	N-R	N-R	IP
3	N-R	N-R	N-R	F-F
4	N-R	N-R	IP	N-R
5	N-R	N-R	IP	IP
6	N-R	N-R	IP	F-F
..	..	...	..	...
..	..	...	..	...
81	F-F	F-F	F-F	F-F

Notice that this point of view treats NR as a special case of GT[6].

[3]the just-calculated Newton-Raphson bargained price

[4]its initial price, i.e., at the start of this iteration

[5]the face-to-face partner's price, i.e., the neighbouring element face

[6]where the first combination in the table is invariably chosen

endplay: Obtaining the first NR iteration defines the end of that particular *play* of the game - this corresponds to the usual single iteration loop. The end of the whole game occurs when adjacent voltages and currents are equal within the specified tolerance.

payoffs: So far the payoff of each outcome is proportional to how well the voltage and current pairs match on all four edges, being largest when the maximum mismatch between currents on each face is smallest. But if the set of choices is greatly expanded as above then a more general notion of payoff needs to be adopted.

Following the economic analogy the most obvious, and perhaps reasonable, payoff is the net profit. This has already been identified with the net power entering the element.

The net profit for the element is thus calculated from

$$profit = \sum_{N,E,W,S} (\chi p)q$$

where the sum is take over all four faces; χ = the exchange rate *(weight)* in the adjacent face; p = the price *(voltage)* on the element's face; and q = the quantity *(current)* entering the element's face.

strategy: In the existing model each element maximises its payoff by reducing all four current mismatches at once. It does this by choosing the NR first-iteration value for all 4 faces simultaneously.

However, with this greatly-expanded range of possible face voltages, it is necessary to choose one of the 81 combinations by a rational method. Once again invoking the economic analogy, it would be reasonable for a bargaining element to choose on the basis of maximising its profit.

So what will be called the *profit method* means that at each play of the game, each element computes the profit it would gain if it used each one of the 81 combinations above. The element then uses that combination

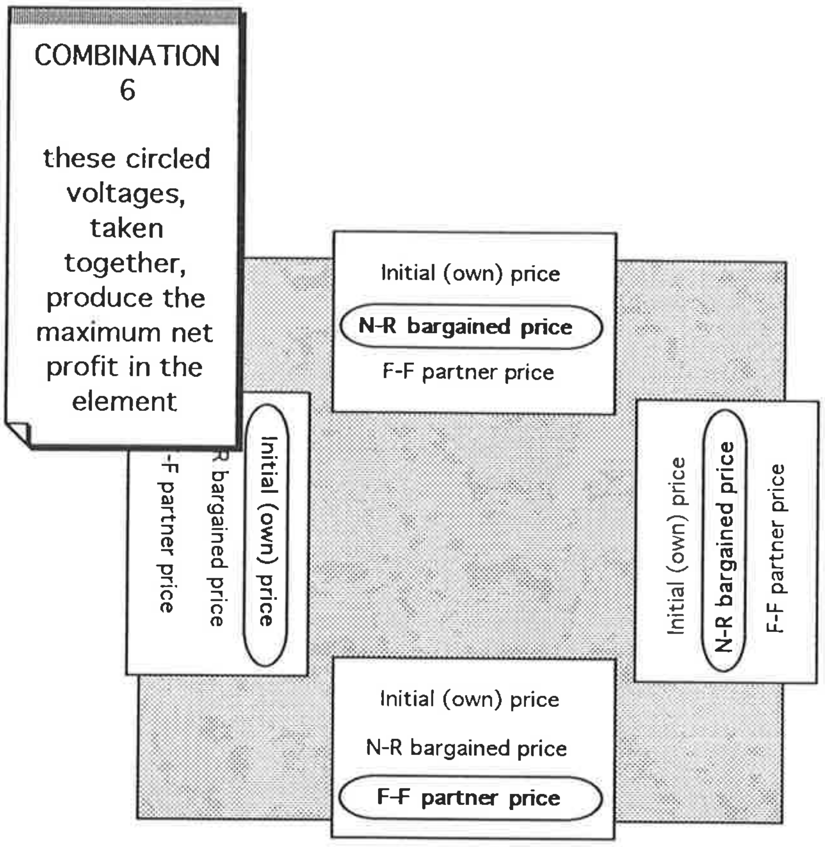


Figure 11.5: ILLUSTRATING PROFIT CRITERION AT COMBINATION 6

of voltages which produces the maximum profit as the basis for starting the next iteration.

For example, as illustrated in figure 11.5 on page 179, if after calculating the 4 bargained prices for the four faces, it turns out that combination 6, namely

North Face	N-R bargained price
East Face	N-R bargained price
West Face	Initially-Proposed price
South Face	Face-Face partner's price

produced the maximum net profit out of all the 81 combinations, then those values of face voltages would be used as the starting-point for the next iteration. That is, the NR values are not always used.

Thus it is seen that the Newton-Raphson iteration technique applied to the convergence of the array simulator is a special case of a game theory method, and the first NR value can be used as a good starting point in the more sophisticated game theory algorithm discussed above.

11.4 Experimental Results

Since NR is a basic game theory method, this investigation distinguishes two cases by different nomenclature, thus:

the smooth method where the NR price, reached by a process of bargaining without convergence, is accepted straight away as the final price for each face - this is how the model is already run, *i.e.* in the previous chapter where the voltages are smoothed over the last 6 iterations.

the profit method where the NR price is reached by bargaining without convergence, and then the criterion of maximum profit is applied to determine whether this NR price will be used or whether the element will stay with its own initially-proposed price or use its face-to-face partner's price on each face for this play. Only voltages at the current iteration are considered.

It needs emphasizing that if combination 1[7] is invariably chosen (the *bargain method*) then this is the method already investigated. The *profit method* thus generalizes the *bargain method* .

The model described above was set up and run, with interesting results.

11.5 Effect of Frequency of Operation

When the *profit method* is applied at each iteration, the system improves for a few iterations and then it locks up, and no further convergence occurs. This seems an unpromising start for game theory, and forces the observation that two extreme behaviours are observed at the outset:

[7]in the combination table, earlier (page 177)

- if the *profit method* always operates[8] then the system locks up after a few iterations and does not converge; this produces what might be regarded as an infinite number of iterations.
- if the *profit method* never operates then the situation remains that of the Newton-Raphson bargaining technique which has already been investigated - the *bargain method* - which is known to converge.

The question arises: what happens in between these two extremes?

To find out, the system was then run so that the *profit method* was only applied every second iteration instead of every iteration, with the default *bargain method* applying otherwise. Once again, however, the system improved for a few iterations and then it oscillated widely without further convergence. This seems very unpromising indeed.

At last, however, the system was run so that the *profit method* was only applied every third iteration instead of every iteration, with the default *bargain method* applying otherwise, as before. This time, by contrast and quite surprisingly, the system showed very fast convergence. In fact it is found that applying the *profit method* every 3rd, 4th, . . . *etc.* . . iteration caused convergence to occur in the usual way.

The results are shown in figure 11.6 on page 182. From this it can be seen that after a frequency of application of every 3rd iteration the rate of convergence[9] varies within a narrow range with a minimum of 78 iterations occurring at a frequency of every 20th iteration. From then on the number of iterations to convergence gradually increases until reaching the *bargain method* value of 419 iterations, where it stays.

[8]i.e., at every iteration

[9]measured by the number of iterations

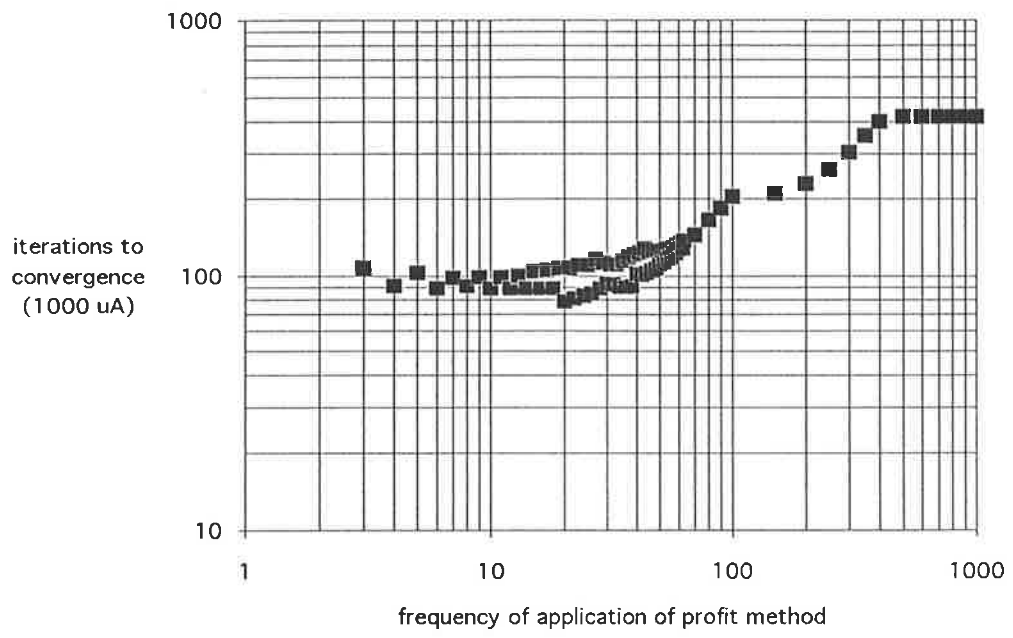


Figure 11.6: EFFECT OF FREQUENCY OF APPLICATION OF GAME THEORY

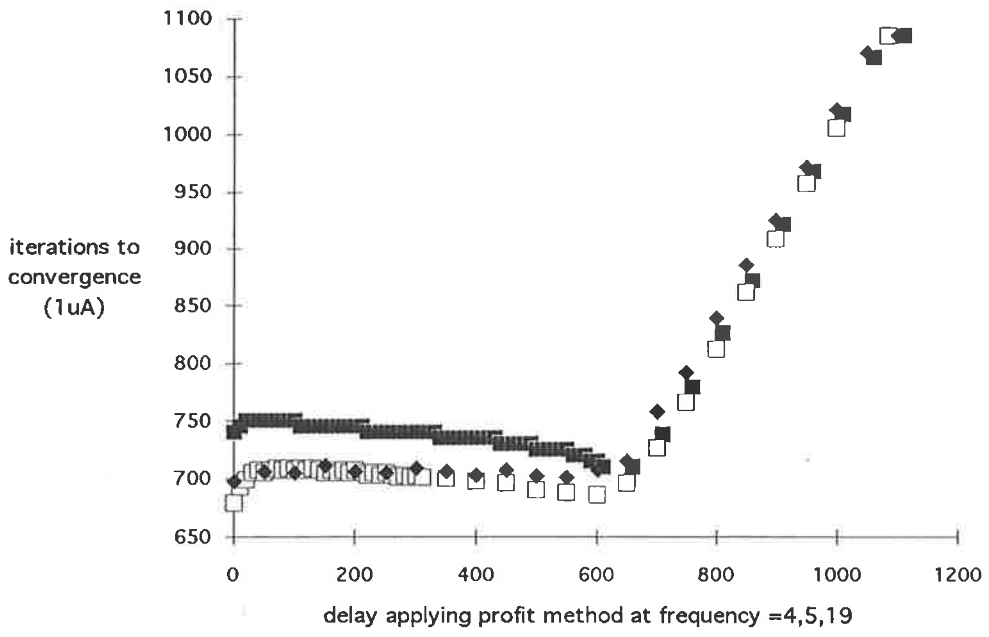
11.6 Effect of Range of Choices

It is found that these results are practically unchanged if, instead of the 81 combinations resulting from 3 price choices on each face, only the 16 combinations resulting from 2 price choices are allowed: the bargained price and the initial price. This is correspondingly faster to simulate, and all subsequent investigations use this reduced range of choices.

11.7 Effect of Delaying Profit Criterion

To see what happens when the *profit method* is delayed, this next study allows the *bargain method* to operate for a while, only later commencing the *profit method* . The results are shown in figure 11.7 on page 183.

The upper plot shows that, at the tight convergence criterion, there is some small advantage in delaying the operation of the *profit method* , but since it is not known how many iterations it will take to converge before the simulation is carried out, and thus how long to delay, then this *post hoc* advantage is totally worthless in practice.



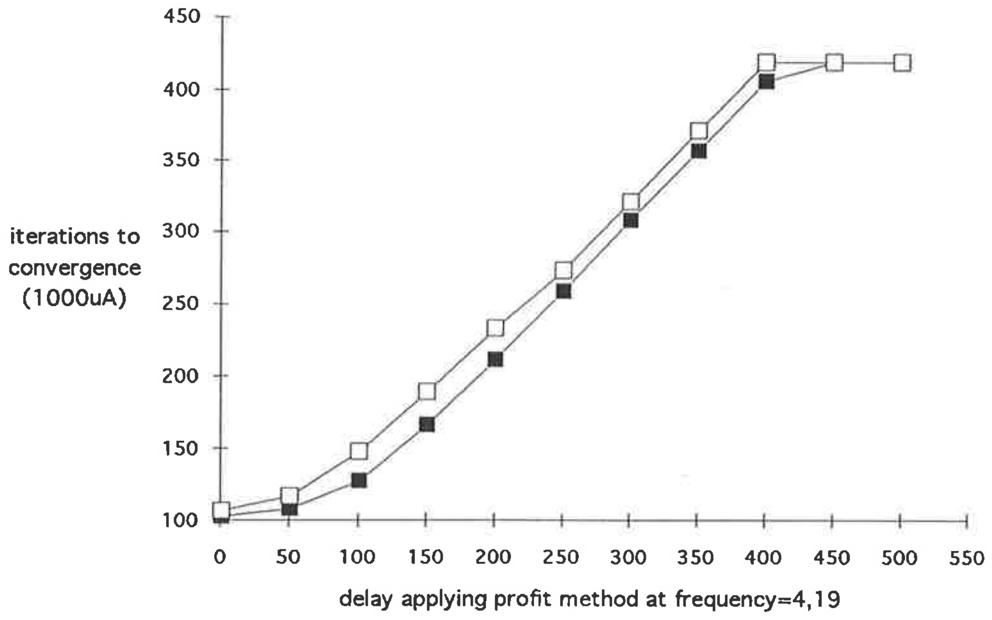


Figure 11.7: Effect of Delaying the Application of the Profit Method

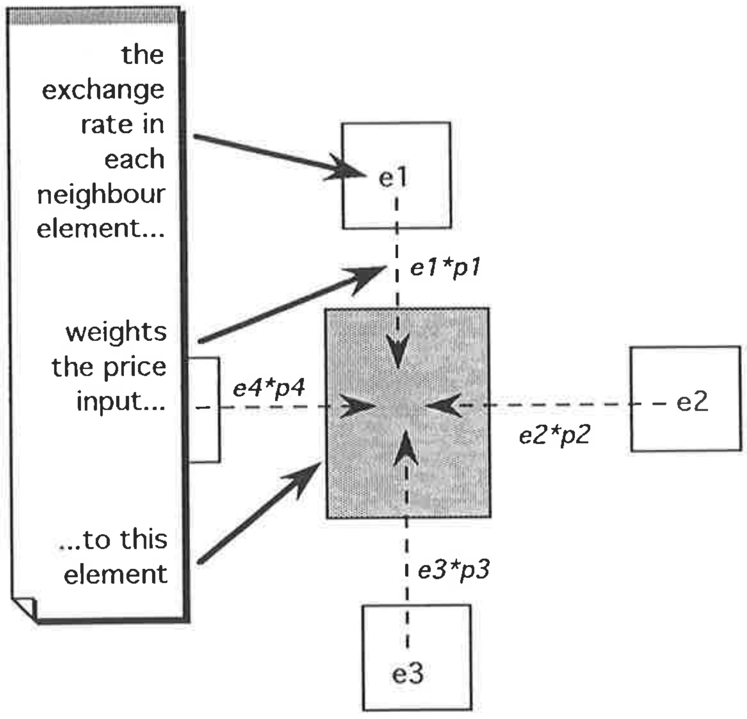

the exchange rate in each neighbour element...
weights the price input...
...to this element
e1
e1*p1
e2
e2*p2
e3
e3*p3
e4*p4

Figure 11.8: MEANING AND OPERATION OF EXCHANGE RATES

In any case, the lower plot shows that, at the looser[10] convergence criterion there is no advantage at all in delaying, and that it is best to start the *profit method* from the very beginning of the simulation.

11.8 Effect of Exchange Rates

Since the exchange rate parameters are available for variation it is natural to study whether they make any difference to the convergence speed.

The meaning of the exchange rates and their mode of operation are both illustrated in figure 11.8 on page 184. The default exchange rate value for all 8×8 elements is 1.

The study starts with a given case: 1000 μA convergence criterion and applying the *profit method* every 5th iteration. Then for each processing element in turn, the exchange rate is varied from its nominal value of 1 and

[10]and probably more commonly-used

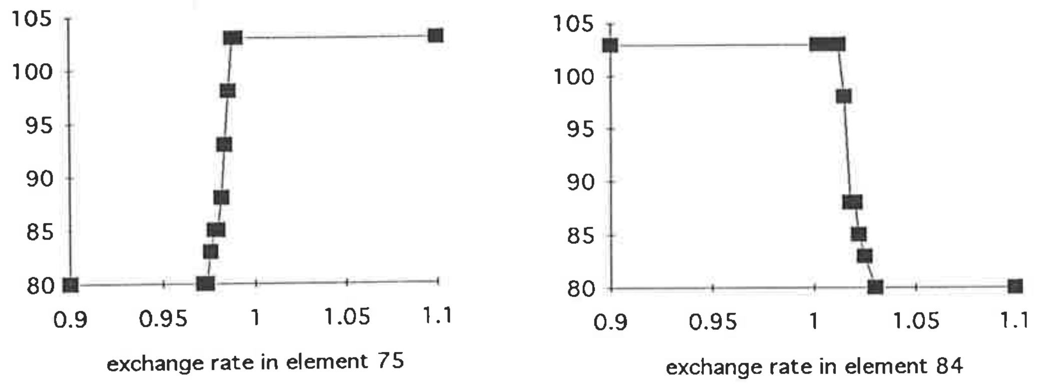


Figure 11.9: EFFECT OF VARYING EXCHANGE RATES ABOUT NOMINAL VALUES

the number of iterations to convergence recorded.

The results for two elements of the array, one in row 7 column 5, the other in row 8 column 4, being representative of the behaviour of all elements, are shown in figure 11.9 on page 185.

The graphs show that as the exchange rate passes somewhere near the nominal value[11] the convergence rate of the array changes fairly abruptly in a step, from one stable value to another, for better or for worse. In fact, simulation at just the three exchange rate values of 0, 1 and 2 produces all the information that can be got about the effect of exchange rate variation.

When the values of exchange rate that produce, individually, the fastest convergence are tried all at once, sometimes a slightly faster iteration is achieved but mostly it is not - this is a very complex non-linear system.

For example, if the rates are changed as shown below then the number of iterations is 78 rather than the individual minimum of 80; but at the tighter 1 μA convergence criterion the analogous case produces an increase rather

[11]default=1

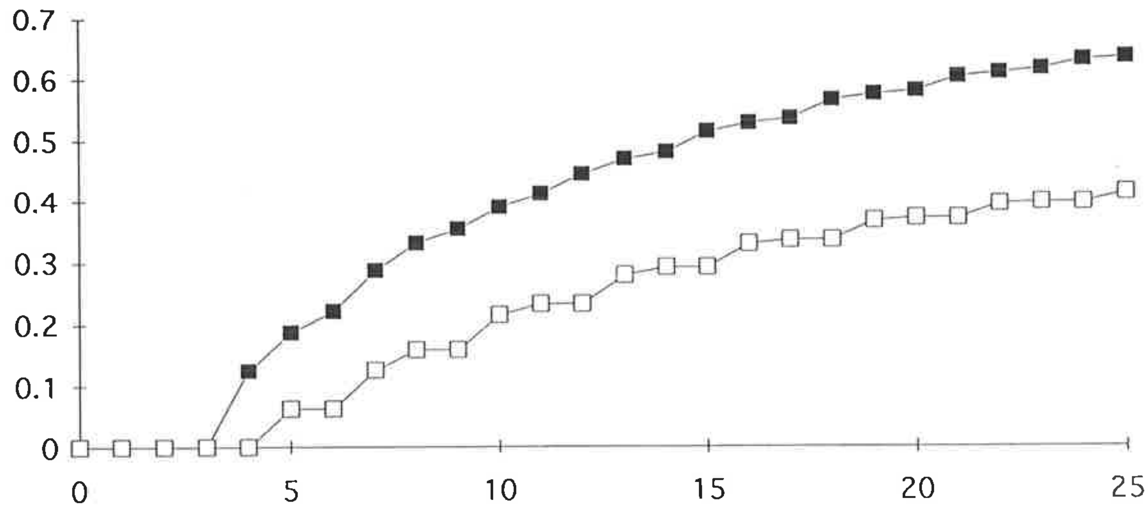


Figure 11.10: PROFIT METHOD 1 × 3 STARTUP NODE VOLTAGES

than a new minimum, as shown below.

$$
\text{standard, 141 iterations} \quad
\begin{matrix}
1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1
\end{matrix}
\longrightarrow
\begin{matrix}
1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1\\
1&0&1&1&1&1&1&1\\
1&1&1&1&1&1&1&1\\
1&1&0&1&1&1&1&1\\
1&2&1&1&0&1&1&1\\
1&1&1&2&1&1&1&1
\end{matrix}
\quad \text{fastest, 78 iterations}
$$

11.9 Convergence Behaviour

In contrast with the original *bargain method* the voltages on certain nodes now have no zig-zag component. This is because the values on each face are being determined by taking into account what is happening on the other faces of the same element to some extent, rather than solving them independently. This is seen in figure 11.10 on page 186. In addition, it is noticeable that the maximum residual, shown in figure 11.11 on page 187 reduces more quickly in the early stage than the *bargain method* but thereafter reduces more slowly. This is a general feature of the method, as will be seen later in this chapter.

11.9.1 Startup Scatter

As in the *bargain+smoothing* case, fascinating results are obtained from a statistical study of the startup phase of the simulator.

Some 800 random starting voltage sets were run to convergence at both

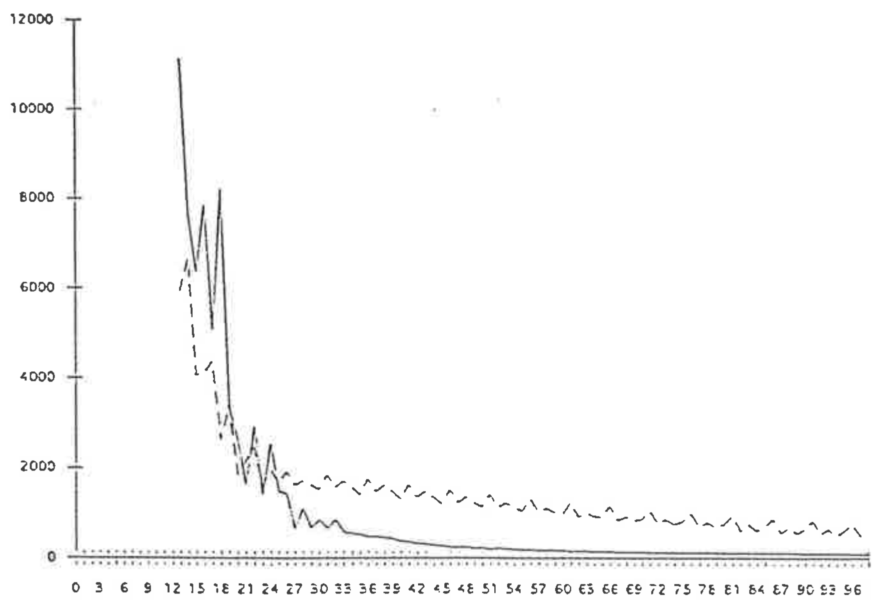


Figure 11.11: PROFIT METHOD MAX. RESIDUAL (DOTTED) COMPARED WITH SMOOTH METHOD (SOLID)

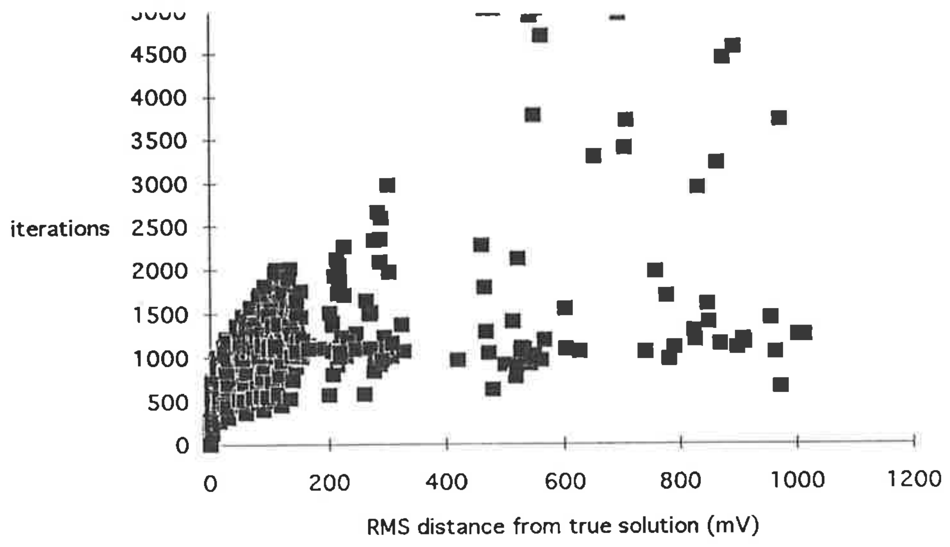
the 1μA and the 1000μA criteria, using a 1×5 game theory frequency algorithm. The results of these experiments are shown in figure 11.12 on page 188 and in figure 11.13 on page 189.

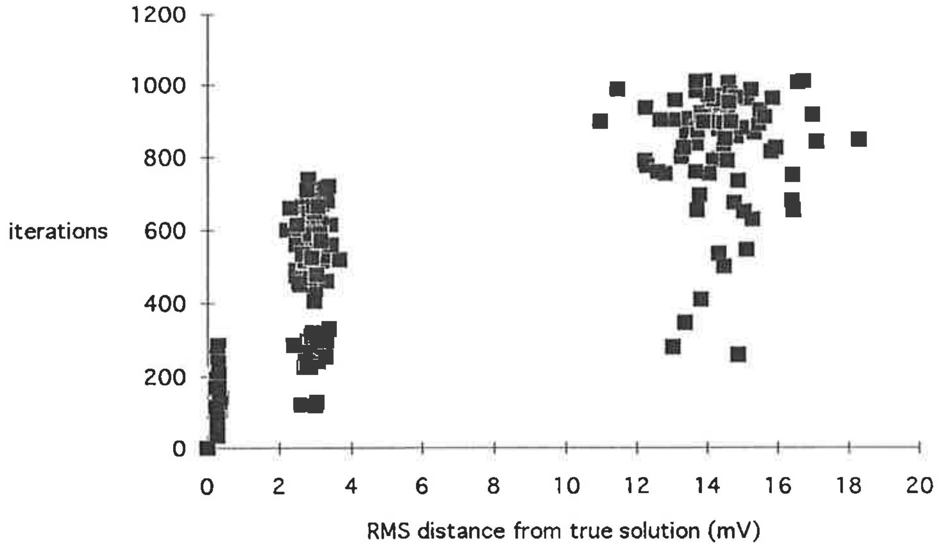
It does not seem to matter how far the system starts from the true solution, it still converges and saturates. But if it starts too far away[12] then it can converge into the 'flip' state instead of the 'flop' state.

Thus there seems no point in trying to get 'close' to the final solution before iterating. Look at the region in the figure say 2-2.5 mV from the true solution. In that small spread of distances, get an immense spread of iterations - from 106 up to 740. Most of the rapid increase takes place very close to the origin, especially in spread of iterations, as measured by the standard deviation. Changing driving voltages will always be making the distance greater than 1mV, unless an impractically-large number of timesteps is used to counteract this.

These results are really saying that the *profit method* algorithm is a very

[12]e.g., if all voltage nodes are initially zero





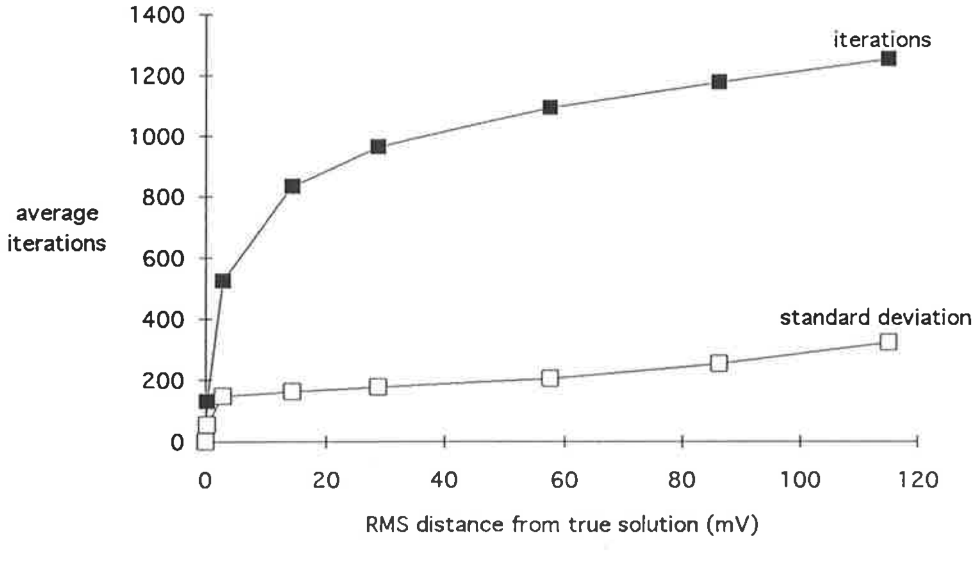
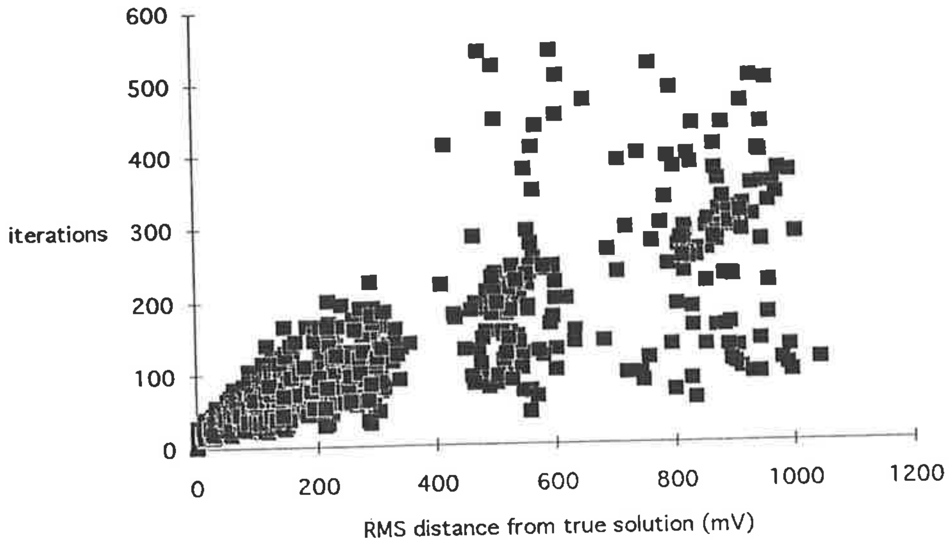
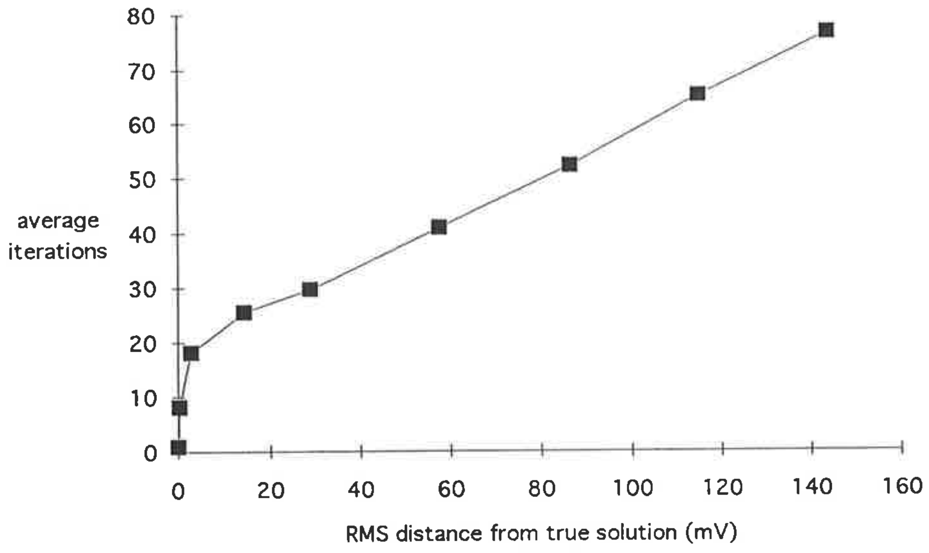


Figure 11.12: Profit Method 1×5 Startup - 1μA convergence





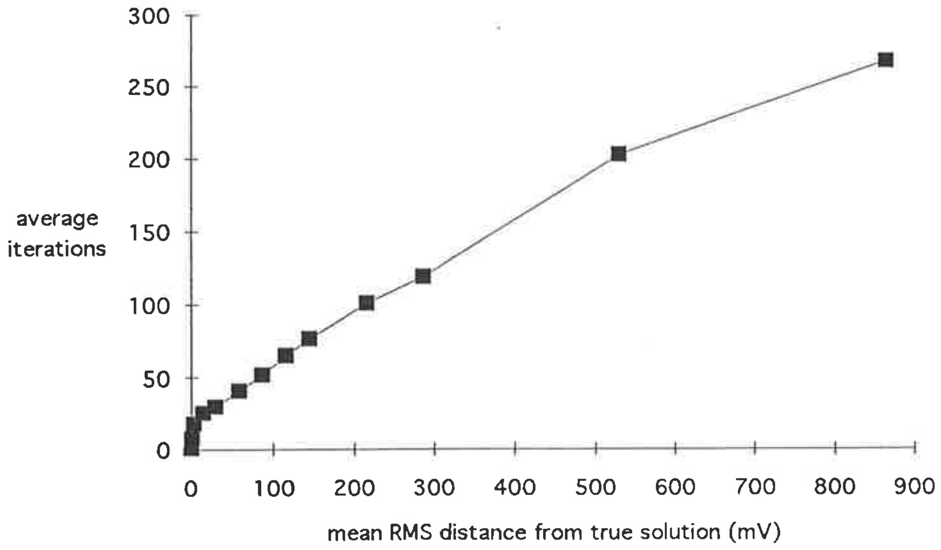
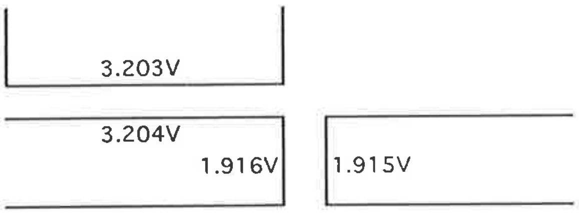


Figure 11.13: Profit Method 1×5 Startup - 1000μA convergence

sensitive beast indeed[13]. For consider: if the voltage at each face differs from its neighbour by as little as 1 milliVolt, like the illustration below of part of the array, then the RMS voltage difference over the whole array is only 1mV,



but depending on just how this 1mV difference is distributed at each face over the array the system can take anything from as little as 1 iteration to as much as 500 iterations - an incredible difference!

11.9.2 Accuracy *versus* Iterations

A study was done over a wide range of convergence criteria to see the effect of two important characteristics: the number of iterations to convergence, and the final accuracy of the solution[14], both of which rise as the convergence criterion becomes more stringent.

The results in the log/log plot in figure 11.14 on page 191 show the nature of this variation for a 1×5 *profit method.* In this range of criteria, an increase in accuracy is paid for by a proportionate increase in iterations.

11.10 Effect of Bits on Accuracy

Up to now the studies have been carried out at the maximum accuracy of 53 bits. Since all aritmetic operations are masked in the C program, studies can be done for any number of bits from 1-53.

[13]certainly as regards initial conditions

[14]as measured by the RMS voltage error

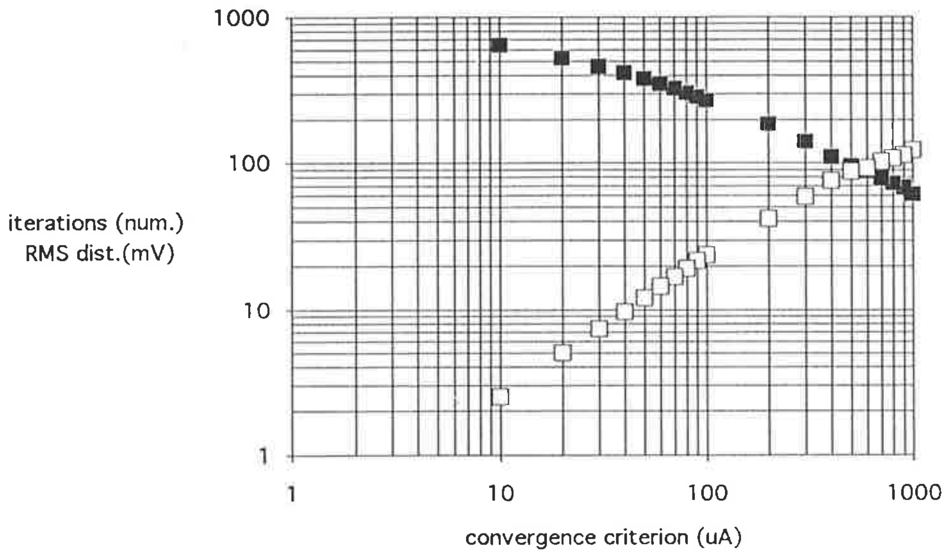


Figure 11.14: 1×5 PROFIT METHOD: ITERATIONS (BLACK) AND FINAL ACCURACY (WHITE)

11.10.1 Convergence at Fixed Criterion

The 1×5 *profit* algorithm starts to fail at about 13 mantissa bits. The results in figure 11.15 on page 192 show that at various convergence criteria the distance of the whole array from the true solution[15] is satisfactory from 53 bits down to 14 bits, below which it begins to diverge seriously from the true solution.

11.10.2 Critical Number of Bits

Consider now the number of bits required for a given accuracy, by setting the number of bits and then seeing how close the system can come to matching the face currents at each node.

For a given startup voltage set[16] the *smooth* and the *profit* method were run with mantissa bits varied in the interesting range of 8 to 24 bits. After the system has settled down to whatever long-term behaviour it can get out

[15]as measured by the RMS voltage error

[16]the same one used in all these simulations

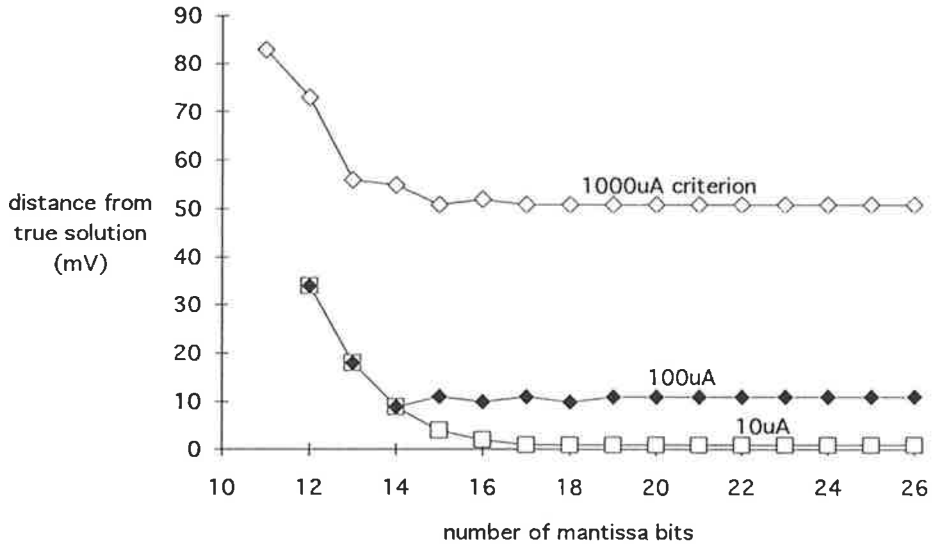


Figure 11.15: Accuracy of Profit Method with Mantissa Bits

of that number of bits, (5000 maximum iterations allowed) we see how close each method comes to a complete match of face currents. Of course, they did not converge when very few bits are used. But at some stage of the iteration process there is a *smallest maximum residual* which measures how well the system matches currents (given that the voltages match perfectly) over all the processing elements. This *smallest maximum residual* was recorded.

The two cases examined are:

smooth which is the *bargain* + 6-point LSQ smoothing;

profit which is the *bargain* + 1×5 profit criterion.

The results are summarized in figure 11.16 on page 193.

Now, these results are amazing. They show that the *profit method* is consistently better than the *smooth method*. That is, given a level of maximum current mismatch[17] to be achieved, the *profit method* achieves it with one whole mantissa bit less than the *smooth method*.

[17]measured by the maximum residual

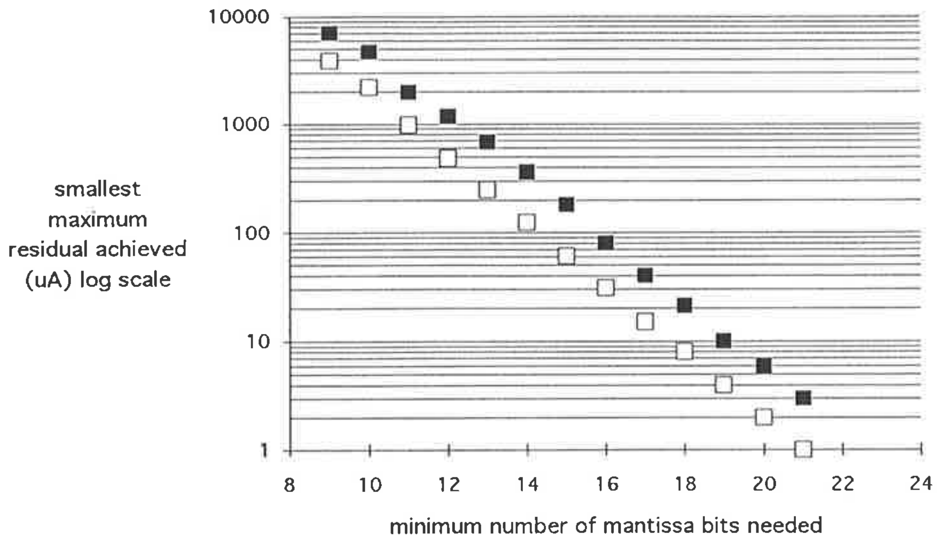


Figure 11.16: Critical Bits - Profit (white) and Smooth (black)

To offset this it takes more operations. This is shown next in a study of the convergence at various criteria producing the total arithmetic operations per timestep.

It is worth noting that when the original optimal lookahead is tried, the results are actually worse than for no lookahead at all - i.e., each bit-set run has its own optimum LA set.

11.11 Operation Count

In contrast to the *smooth* method, the *profit* method produces, over the range of 12 to 53 mantissa bits, and over a wide range of random profit criterion frequencies, the characteristics for the number of arithmetic operations summarized in the following table.

Operation	Mean Number per Iteration	Range
multiplication	860	593-1281
division	530	375-765
addition	500	312-671
subtraction	250	187-375

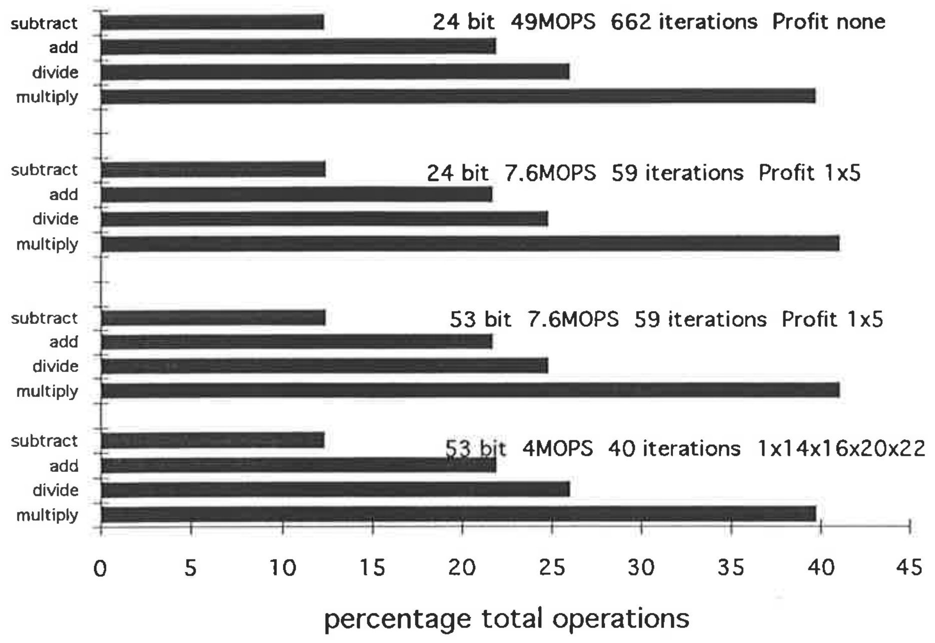


Figure 11.17: Profit Method: Mean Arithmetic Operation Profile

The reason for the large spread is that the more frequently the profit criterion is applied the more calculations are needed overall, and so the greater mean number of operations are required per iteration. The incidence of that varies with the accuracy required.

11.11.1 Operation Profiles

The mix of operations for the 1000μA convergence criterion is summarized in the operation profile shown in figure 11.17 on page 194. The profiles hardly change with number of bits and frequency of application, even though the cases represented range from 40 - 662 iterations, and from 4 - 49 million operations overall from the startup situation.

By way of comparison, the full set of operations carried out by the processor array is counted and presented in the set of profiles shown in figure 11.18 on page 195. Once again the *profit method* shows a steady profile over a range of mantissa bits and frequencies, whereas the *smooth method* shown a vast increase in absolute value operations as it attempts to work at very high

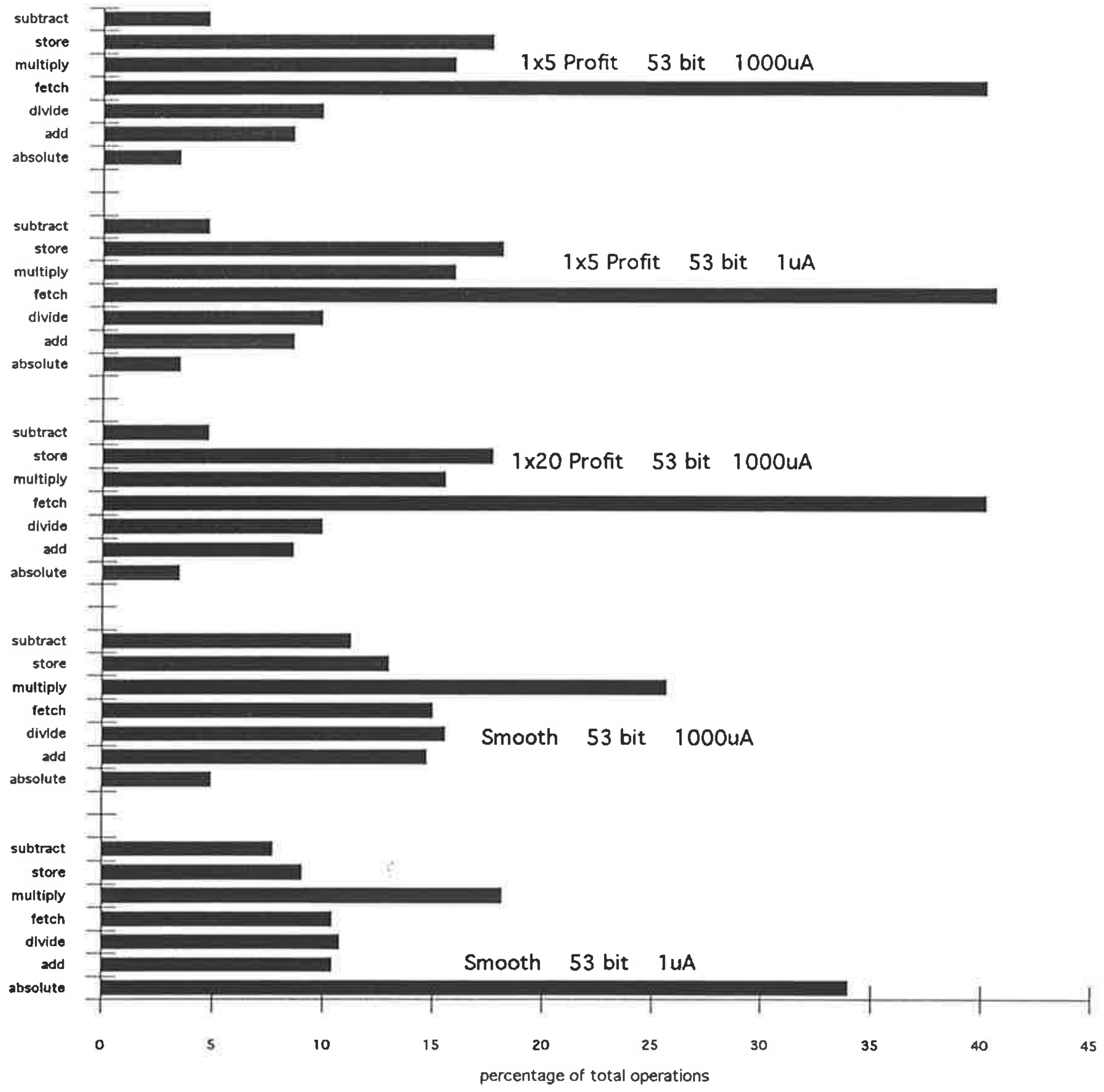


Figure 11.18: OPERATION PROFILES: COMPARISON BETWEEN VARIOUS STRATEGIES

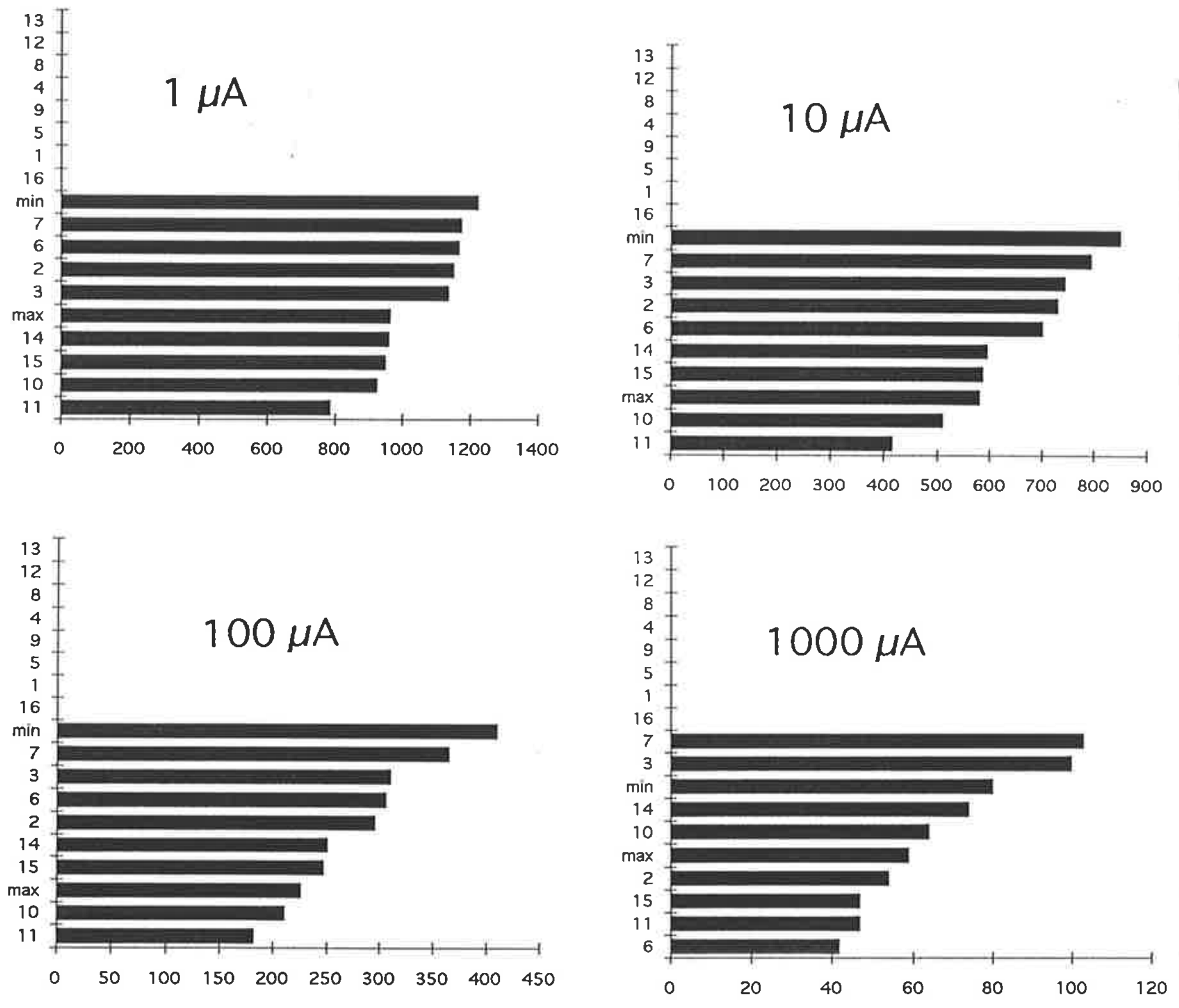


Figure 11.19: EFFECT OF DIFFERENT CRITERIA ON NUMBER OFITERATIONS

accuracies.

11.11.2 Combinations *vs.* Profit Criterion

To check that the *maximum profit* criterion is indeed the best one, a series of tests carried out using all the different criteria produced the results summarized in the iterations profiles shown in figure 11.19 on page 196. For the four convergence criteria of 1μA, 10μA, 100μA, 1000μA these show that the *maximum* profit criterion is consistently better than the *minimum* profit criterion. Using any one of the 1 thru 16 combinations (from the $2 \times 2 \times 2 \times 2$

choices) to start the next iteration produces variable results which generally depend on the connectivity of each circuit.

11.12 Multiple Frequency Effects

Given the strange behaviour of the *profit method* criterion application frequency - that it fails to converge if applied too frequently, and that a vague minimum exists - it is natural to wonder how to find the optimum frequency before simulating a circuit.

It has already been seen that if the *profit criterion* is applied too often[18] no system convergence results, and a minimum number of iterations is achievable[19]. But this is determined after the event, for this particular circuit and for these startup conditions. There seems to be no general method for predicting which frequency is best in advance.

The question then arises: in default of a method for this, is it possible that a frequency chosen at random would work better than doing nothing? To answer this, a range of application tactics are now examined.

The previous study starts with the simplest case of pure frequencies. But more generally a set of multiple frequencies needs to be considered. It will be seen that this simulates the effect of random application of the profit criterion.

11.12.1 Individual Random Extremes

Consider the set of profiles in figure 11.20 on page 198. For every 100 iterations, the vertical lines show exactly when the profit criterion is to be applied. The application points appear randomly distributed, but this is really the result of superimposing a few pure frequencies on top of one another. These figures summarize the results of applying the *profit criterion* at various

[18]every iteration or every second iteration

[19]at around a frequency of every 19 iterations

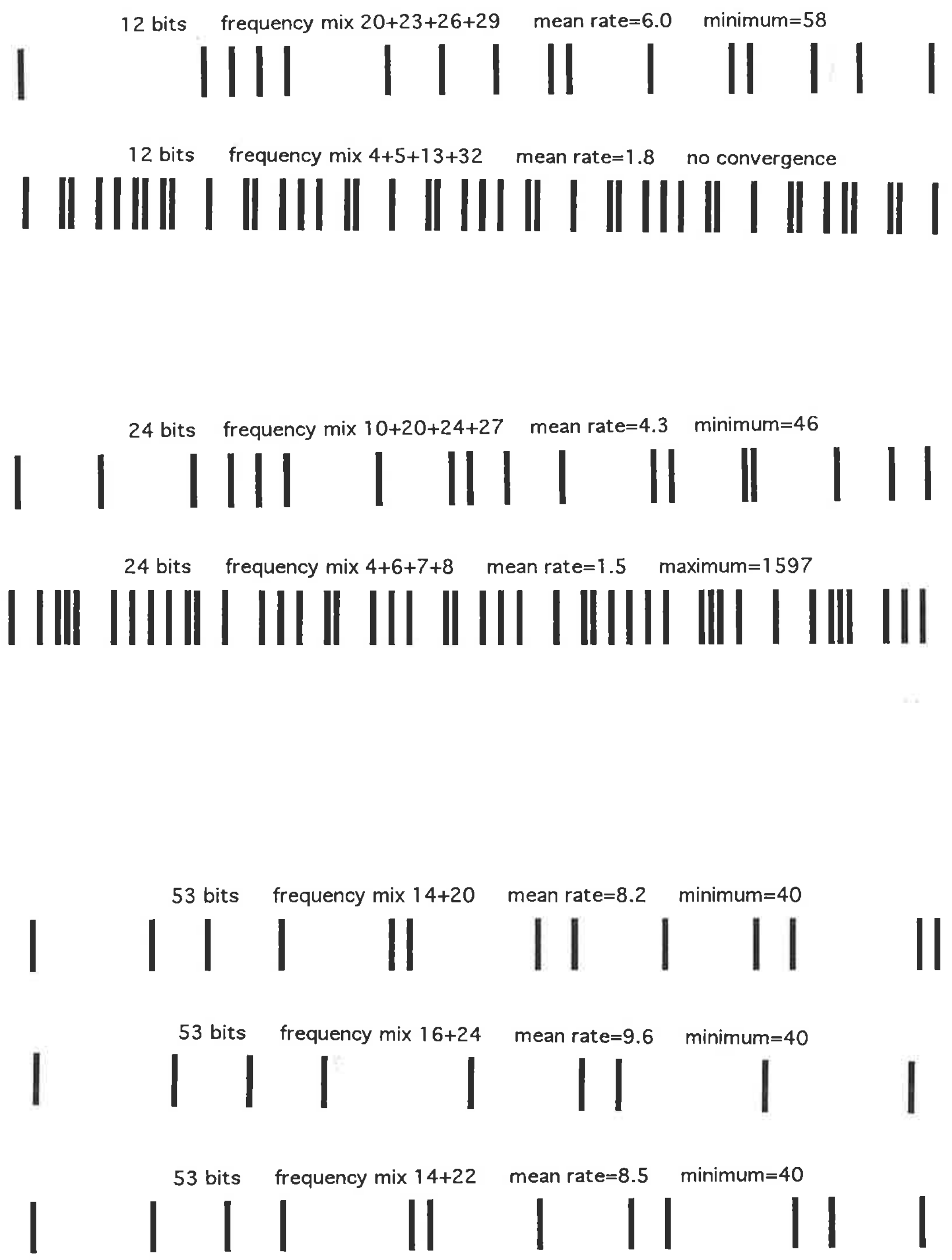


Figure 11.20: PROFIT METHOD: PSEUDO-RANDOM APPLICATION FREQUENCIES

frequencies and using various numbers of mantissa bits.

The results are somewhat surprising but suggest some predictabilility.

The top 2 profiles for 12 bits show that quite extreme behaviour can be produced. If the profit criterion is applied every 20th, 23rd, 26th and 29th iteration then the system converges with the least iterations; whereas if the profit criterion is applied every 4th, 5th, 13th and 32nd iteration then no convergence of the system occurs, effectively producing an infinite number of iterations. The mean frequency for the least iterations is once every 6.0 iterations, whereas the mean frequency for the convergence failure is once every 1.8 iterations. Comparing this with the results already found for the case of pure frequencies[20] shown in figure 11.6 on page 182, in it is seen that the mean frequency rate is not inconsistent with that result.

To emphasize that this is not entirely predictable, however, the next 2 profiles show, for 24 bits, that the minimum number of iterations is obtained at a mean frequency mix of every 4.3 iterations; whereas a maximum *but not a failure of convergence* is obtained at a mean frequency mix of every 1.5 iterations, which is less than the 2 iterations that produces failure in the pure frequency case. The bottom 3 profiles are all different but all result in the absolute minimum number of iterations found by experiment.

So for these mixed frequency applications of the profit criterion the result cannot be predicted from just the individual frequencies or from the mean frequency, although there is the strong indication that if the mean frequency is less than about 2 or so there is danger of convergence failure. Thus it appears that to get onto firmer ground a more systematic statistical study is necessary, and this is reported next.

11.12.2 Mean Random Scatter Study

In order to assess the viability of choosing a set of frequencies at random, a large number of simulations were run, each allowing a mix of 4 randomly-

[20]where failure to converge occurs at frequencies of 1 and 2 iterations

chosen frequencies, all starting at the first iteration. The results for the two practical extremes of 12 bit and 24 bit accuracy at the 1000μA convergence criterion are shown in the top and bottom of figure 11.21 on page 201. The top graph[21] shows that when the system takes, say, 52 iterations to converge, the number of processor operations[22] required varies from 57 million up to 78 million per simulationdepending on the precise frequency mix.

The lower plot[23] shows a similar spread, but there is also the possibility of getting a frequency mix that takes a very long time to converge.

For the mid-range case in the bottom plot, it is found that a line of best fit[24] through the data points yields the approximate relationship: *millions of operations per simulation* $= (\frac{iterations}{5} - 3)$; this might be useful for estimating the total time taken by the array simulator for a particular architecture.

Now, it has already been seen that for a pure frequency of every 5 the system produces 59 iterations and 7.6 million operations for startup. So shown in figure 11.21 on page 201, that choosing a set at random, compared with this, produces[25] an average number of 54 iterations. Since this is less than the 59 iterations above, choosing a random set could be better than having no frequency, but not by much.

To collect a lot of these results together, the array shown in figure 11.22 on page 202. shows the iterations at 1x5 1000μA for a systematic study of separate profit criterion application frequencies, from 1 up to 30. The results are somewhat unclear because there is some aliasing occurring; this means, for example, that the result for frequency=3 contains some part of the result for frequencies=6,9,12 and so on. A chaotic area of relative minima occurs in the middle of the array, which therefore shows that there is a rough degree of correlation between these results and the expected behaviour based on the

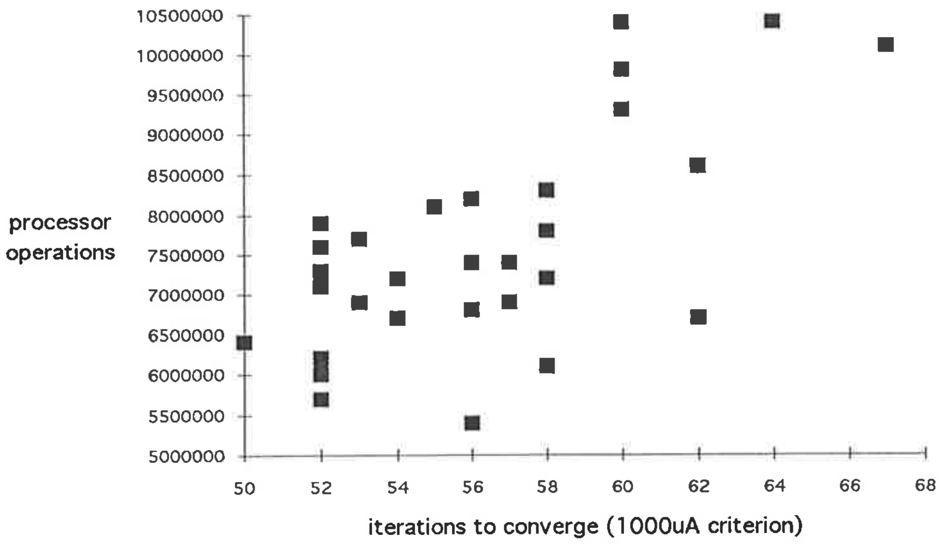
[21]for a 12-bit mantissa

[22]multiplications+divisions+additions+subtractions

[23]for a 24-bit mantissa

[24]in the least squares sense

[25]after removal of frequencies near every 1 and 2 iterations



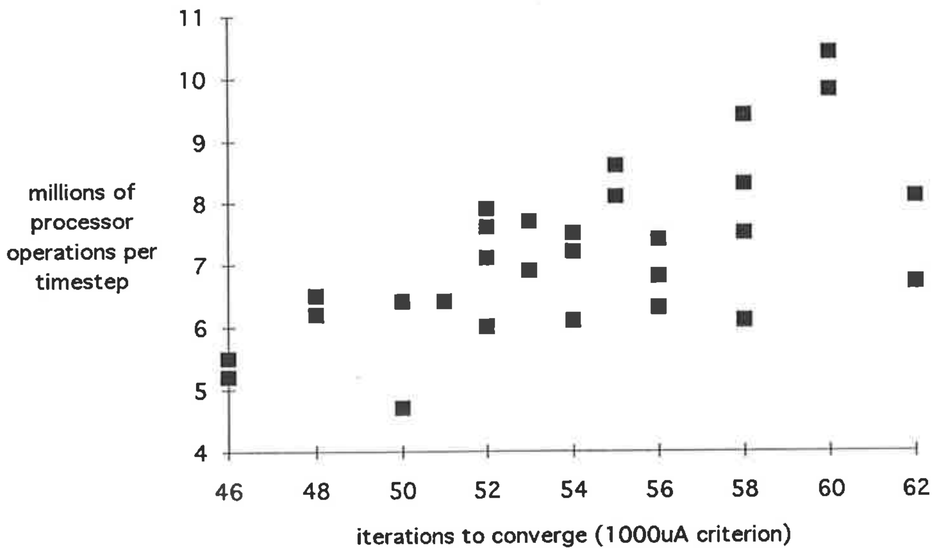


Figure 11.21: RANDOM PROFIT CRITERION APPLICATION - 12 BITS (TOP) & 24 BITS (BOTTOM)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1																														
2																														
3			67																											
4			113	52																										
5			70	72	59																									
6			67	60	59	46																								
7			70	60	56	53	60																							
8			68	52	54	52	53	46																						
9			67	60	59	58	53	58	64																					
10			70	56	59	54	50	48	58	46																				
11			67	52	56	52	50	52	49	50	63																			
12			67	52	52	46	48	48	58	48	48	44																		
13			67	52	54	52	50	52	49	56	52	49	62																	
14			61	52	54	48	60	48	58	44	48	48	56	46																
15			67	56	59	58	53	46	46	62	45	62	49	51	61															
16			68	52	52	46	53	46	46	46	52	48	56	48	53	52														
17			67	52	59	48	53	46	46	46	48	48	49	48	46	52	63													
18			67	52	54	46	48	46	64	46	48	44	56	48	62	48	55	50												
19			67	52	56	48	50	48	51	48	48	48	45	50	44	46	48	55	61											
20			67	52	59	48	48	46	46	46	48	48	52	40	59	48	48	48	57	51										
21			67	52	54	48	60	48	46	46	46	52	52	46	46	46	46	48	48	57	65									
22			67	52	51	48	48	48	49	48	63	44	52	40	44	42	44	44	48	50	59	53								
23			67	52	54	52	50	52	54	44	43	42	56	46	56	52	50	50	50	50	52	57	67							
24			67	52	52	46	53	46	52	46	52	44	52	46	44	40	44	42	46	46	48	52	61	55						
25			67	56	59	48	60	46	51	44	43	54	54	46	46	54	55	54	54	54	54	54	54	59	69					
26			67	52	54	46	53	46	53	44	52	44	62	46	44	42	44	42	44	44	48	50	50	56	63	57				
27			67	52	59	48	53	46	64	46	43	52	43	46	42	52	50	58	54	54	58	52	54	54	58	61	71			
28			61	52	62	46	60	44	44	44	52	44	51	46	53	46	44	42	46	44	46	48	50	52	54	60	65	61		
29			64	52	59	48	50	46	51	44	45	52	45	60	53	44	44	55	52	54	56	54	56	54	58	58	62	65	73	
30			67	52	59	46	53	44	46	46	45	46	51	46	61	49	46	44	46	44	46	48	50	52	54	56	58	62	67	65
31			64	52	59	48	53	46	51	54	45	52	45	54	49	52	44	48	48	52	52	54	56	54	56	56	62	62	64	64
32			68	52	59	46	53	46	46	44	43	46	49	46	53	52	55	46	46	46	46	48	50	52	54	56	58	60	62	66
33			67	52	59	48	50	52	46	50	63	52	45	54	49	60	55	46	46	52	50	54	54	54	54	56	60	60	64	62

first frequency of application of profit criterion

second frequency of application

Figure 11.22: ITERATIONS AT 2 RANDOM PROFIT CRITERION FREQUENCIES

mean frequency of application.

11.12.3 Assessment of Random Criterion

These few heuristic experiments show that this is a very strange phenomenon indeed. If someone recognises what the underlying process is then they could explain it and point the way forward to systematic improvement.

The oddest thing is that if the profit criterion is applied too often, it stops working. The structure of the results previously obtained in figure 11.6 on page 182 shows this.

How can the best application frequency be determined as an adaptive process during simulation? This is a topic for future research. However, it is now known that too frequently inhibits convergence, so there would be no way to allow applications more frequent than every 2 iterations.

Note that here the possibility of *each element* having its *own* set of frequencies - a tactic which might be superior to one using the same value for the whole array – has not been addressed. Once again, future research into adaptive techniques is indicated.

The results are like simulated annealing, already used in simulation of circuits [Rut89], in that it jolts the process out of a bad rut. This *intermittent reinforcement* aspect of the algorithm might have analogues in economics, neural networks, and signal processing, which might shed some light on why it acts like this. The system seems to need a bit of time to allow a different profit criterion value to exert its influence. It is in this way that *game theory* with its application of the profit criterion acts like the economic analogue of simulated annealing. Not until this is understood can a systematic and adaptive use be made of this potentially powerful array processor simulation technique.

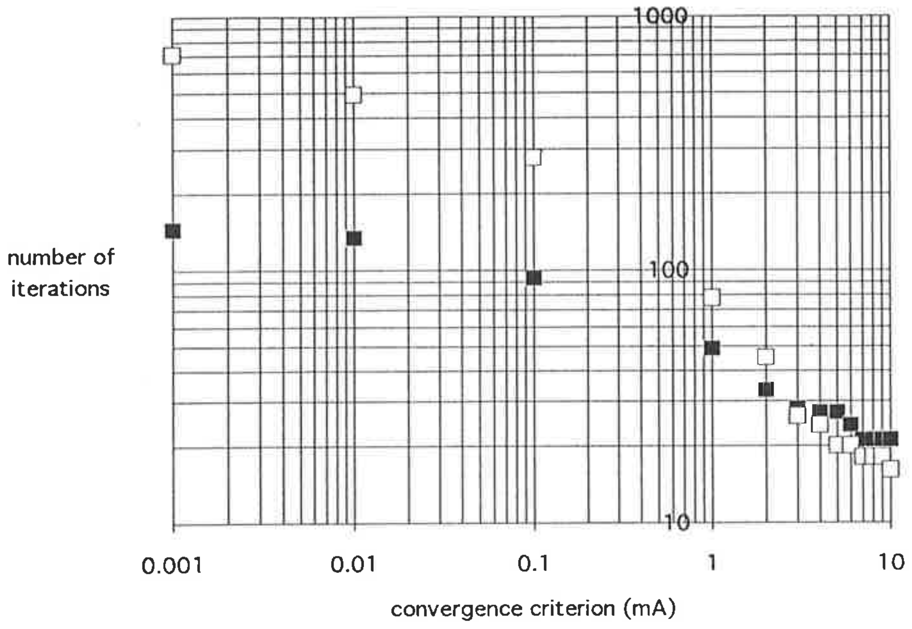


Figure 11.23: COMPARISON OF PROFIT (WHITE) AND SMOOTH (BLACK) STRATEGIES

11.13 Comparison: NR and GT Convergence

The final study is of the behaviour of the Newton-Raphson[26] and the Game Theory[27] over a wide range of convergence criteria. The NR method[28] is compared with the GT method[29], and the results are shown in figure 11.23 on page 204.

This log-log plot shows that the *smooth method* performs better in the region of small convergence criteria, whereas the *profit method* performs better in the region of large convergence criteria.

They seem to perform equally well around the 3 milliamps criterion. To examine the behaviour here in more detail, this region is expanded as shown in figure 11.24 on page 205. It can be seen from this that the advantage of

[26]smooth method

[27]*profit method*

[28]6-point smoothing with lookahead, previously investigated

[29]applied every 5 iterations

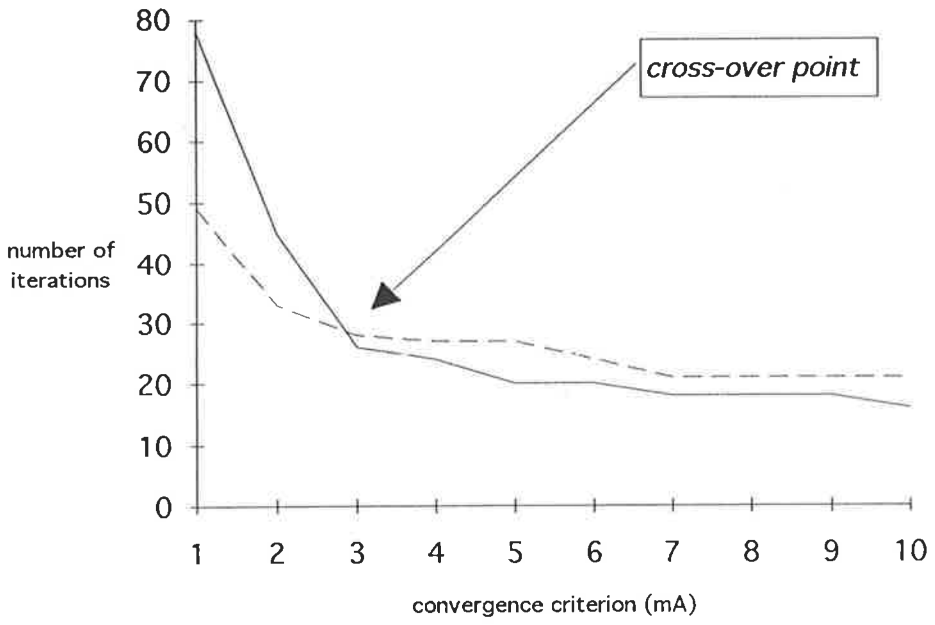


Figure 11.24: WHERE PROFIT (SOLID) PERFORMS BETTER THAN BARGAIN (DOTTED)

the *profit method*, while slight, is definite.

The conclusion from the startup scatter study, consistent with others made already, seems to be that the *profit method* is better suited to loose convergence studies where that accuracy is sufficient.

The assessment of critical bits results indicates that there is a tradeoff for circumstances: smaller processors can get, say, 22 bits down to 16 bits giving, say a 10% area reduction but perhaps a much greater than 10% operation or time increase - and *vice versa* if desired. Better still is some type of cyclic real number representation or one that is adapted to speed/area tradeoffs, like that of Bergamann and Fan [BF90], that would allow accuracy to be chosen without affecting the chip architecture. There is scope here for innovation.

11.14 Significant Conclusions

A new approach has been developed for the array processor. The approach of game theory has been tried because of some suggestive analogies with modelling by the contact method.

The experiments along these lines are very encouraging. The studies here indicate that:

the economic $\Leftrightarrow$ electronic correspondence suggested in figure 11.1 on page 171 and tentatively identified in figure 11.2 on page 174 are justified.

the game theory profit method devised above appears to be validated.

The method actually produces quite a large speedup in convergence. It does this in the early stage of the convergence process, when the mismatch between the current and final solution is greatest but where the plain Newton-Raphson technique does a poor job. It does not produce as much speedup in the latter stages of the convergence process where the mismatch is small but NR does better.

For loose convergence criteria the speedup is remarkable; for tight convergence criteria it is still disappointingly slow although present. Whether tight or loose convergence specifications are required depends on the accuracy wanted in the overall estimate of the cumulative probability distribution.

The investigation at different convergence criteria shows that the game theory *profit method* is superior to the game theory *smooth method* in the loose convergence area, i.e., where a rough and ready picture of the spread of operating speeds is wanted.

Since a minimum occurs in the results where parameters can be manipulated to obtain some degree of optimisation[30], it suggests that an adaptive strategy is called for, not only overall but for each element.

[30]e.g., using exchange rates

Part IV
Evaluation

"Why? Because it dares! To dare: progress is the reward of that."

VICTOR HUGO

Chapter 12

Overview of Results

12.1 Introduction

We have looked closely at what happens when a VLSI circuit is simulated and found to be not quite fast enough to meet its designed specifications. Elements along the critical timing path are re-designed so it works faster. However, a problem now arises that has a consequence which is unpredictable if stochastic simulation is avoided. If the design has to run faster, the circuit dissipates more heat since it works more. Then the individual transistors in the redesigned parts of the circuit become slightly larger, and the chip itself gets slightly bigger overall. This means that the fabrication yield drops in a well-understood way. The result is that the design yield has improved, but the fabrication yield has fallen. Since the net yield is the product of these two opposing trends, the result is somewhat uncertain, only being estimable by an enlarged stochastic simulation that takes into account many more factors than are dealt with here. Nothing short of the full cumulative distribution function of finishing times is required.

To obtain this desired cumulative distribution function of finishing times it is necessary to perform some 500 Monte Carlo simulations. The time this takes on a typical workstation inhibits its use. To speed it up, a very fast and very accurate method of stochastic modelling on modern workstations has been sought. This search has seen a new 'best-fit' for the spread of operating times found, a new method for predicting statistics of concatenated structures invented, and some very strange array simulation behaviour discovered. It has ultimately led to a method which is not really fast when it is accurate, and which is not really accurate when it is fast; this trade-off between speed and accuracy is often found in circuit simulators.

Surprisingly, however, an incidental interpretation of the meaning of the Newton-Raphson iteration technique has also been discovered, leading to the study of the application of game theoretical methods to the simulation, and ultimately to the possibility of a new paradigm which bears on the future design of wafer-scale simulation engines.

There is an overall diagram on the next page, to help in orientation, showing all facets of the thesis.

12.2 What has been Done?

A number of approaches to circuit timing simulation are examined in great detail, and these are classified as variants of the bargain method, thus:

bargain: take the first Newton-Raphson solution and then iterate;

smooth: bargain smoothed over last 6 values using least-squares best fit;

lookahead: bargain + smoothing + prediction look-ahead by less that one iteration;

game: formal game theory approach to whole of the bargain method;

profit: bargain method + profit criterion.

12.2.1 Array Model

A layout of transistors in an array, similar to the physical layout of transistors on the leaf cell circuit, is assumed. Each processing element in the array

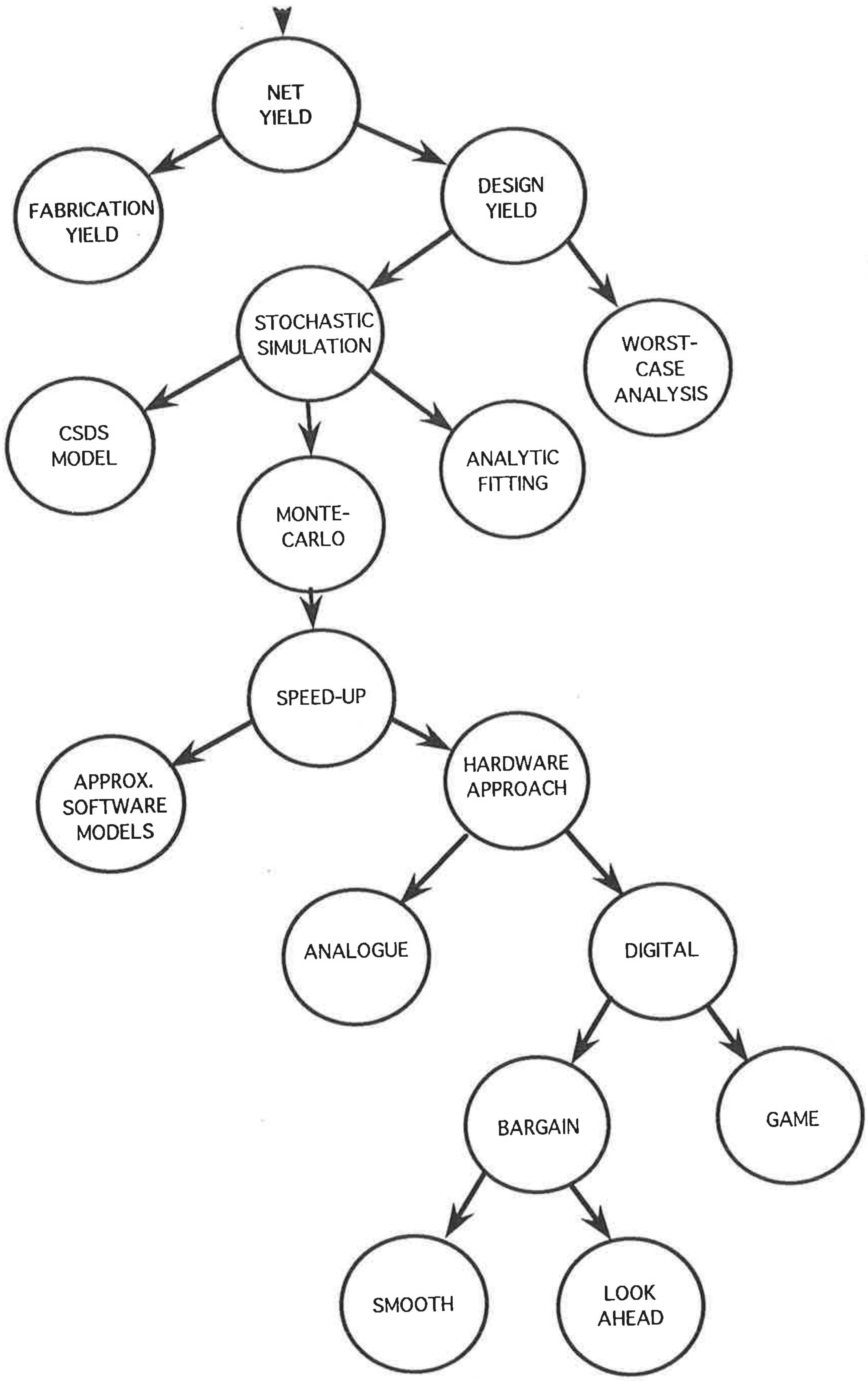


Figure 12.1: MAP OF THE THESIS

contains a mathematical model of the transistor at the node in the physical array. It receives the state of its nearest neighbours and computers its new state in response to this information. Global conditions (such as conservation of current) are catered for by reducing them to local statements of the same principle. Technically, this design is known as a mesh-connected SIMD (Single Instruction Multiple Data) array- processing computer.

12.3 What has been Found?

12.3.1 Probability Distribution

The form of the distribution of finishing times has not previously been discovered. The nearest useful analytic expression is due to Downs, Cook and Rogers [DCR82] who produce a very good approximation by expanding the probability distribution function as an Edgeworth Series

$$f(t) = \frac{K(t)}{\sigma\sqrt{2\pi}} exp[-\frac{1}{2}(\frac{t-\mu}{\sigma})^2]$$

$$K(t) = 1 + \frac{\sqrt{\alpha}}{3!\sigma^3}(t^3 - 3t) + \frac{(\beta - 3)}{4!\sigma^4}(t^4 - 6t^2 + 3) + \frac{10\alpha}{6!\sigma^6}(t^6 - 15t^4 + 45t^3 - 15) + ...$$

where μ=mean, σ^2=variance, α=skewness and β=kurtosis. This Gaussian distribution, shaped by the function $K(t)$, is a suggestive form of Taylor expansion, rather than a serious attempt to find the true distribution. As a result of the work in this thesis, the spread of finishing times is now known to be best described by the Erlang[1] distribution

$$f_{\alpha k}(t) = (\alpha k)^k \, t^{k-1} \, e^{-\alpha k t} / \, \Gamma(k)$$

where $\alpha = \frac{1}{\mu}$ and $k = (\frac{\mu}{\sigma})^2$ are parameters of the distribution. It is not surprising that the Erlang fits so well. It was tried because it is recognised that the process from which it is derived is fundamentally similar to the leaf-cell process: that of passing a signal through a set of inter-related transistors in a leaf cell, compared to that of being served by a set of queues in teletraffic systems, as described by Takacs [Tak62].

[1]or Truncated Gamma

12.3.2 Monte Carlo Methods

An important question is: Are about 500 simulations necessary or are there other methods? Monte-Carlo simulations require many runs but they are good for any initial parameter spread whatsoever. The attempt[2] to reduce the necessary number of Monte Carlo simulations from around 500 down to 1 failed because the resulting cumulative distribution function was not found to be accurate enough. A very accurate model is needed for stochastic simulation; less than one percent error is allowed [WYC87].

12.3.3 Concatenation of Cells

The TIME method, which requires stochastic simulation of only the first two cells of an n-cell concatenated structure, one at a time, was invented. It produces the required statistics with great accuracy, and at vastly reduced cost compared with a full simulation involving the whole n cells. Its accuracy is superior to the rudimentary parameterized block-concatenation scheme of Benkoski [BS87].

12.3.4 New Interpretation of NR Technique

The idea of treating the array as device-based rather than node-based, leads directly to a 'bargaining' situation between 2 *PE*s and hence to a game theoretical implementation. In the traditional method, with each node as the primary focus, it is impossible not to think of solving the equation $f(x) = 0$ and hence automatically use the geometric Newton-Raphson technique. But focussing on 2 *PE*s, each of equal status, immediately demands the possibility that each has its own view of the common node voltage and hence there are two different values to deal with, two current curves to consider, and the question of how to come to an agreement over them becomes of prime concern.

[2]the CSDS model previously examined

12.3.5 Game Theory

It has been shown that Game Theory can be applied to circuit simulation, in its array-processor form. However, although one might not be entirely convinced that the analogy is complete and rigorous, if it is then a tremendous field of new research could open up to application of these methods.

The game theory *profit criterion* algorithm is only fast if relatively low accuracy is required. For the very high accuracy demanded by stochastic simulation it seems that, in its current form, it would be somewhat slow compared with other methods. This suggests that its future applications lie in other fields: still simulation and array-processing, but where fast initial convergence with low accuracy is tolerable - for example, in the neural network area.

As far as the application of further methods is concerned, as early as 1953 D.B. Gillies showed that a sizeable swath of plural games possess cooperative solutions. Buch and Taumann [BT92] deal with bargaining problems having n + 1 players. One player has a special role in the game. He is endowed with a set of actions, each of which dictates a payoff vector for a certain subset of outcomes. Each player gets a higher net payoff in the bargaining solution than in the non-cooperative solution. More on co-operative game theory can be found in the papers of Eichberger [Eic92], in the strategic collaboration analyses of Colman [Col82], and in the co-operation chapters in Shubik [Shu83]. There is also the suggestive experiment by Smith and Williams [SW92] revolving around a system called *a double continous auction* causing much faster convergence in price levels in a system which could be similar to the array studied here.

Initial indications are, however, that technically within the field of economics, the methods pursued here seem to be a subset of a partial general equilibrium system, with a measure of oligopoly; this is reported to be an unsatisfactory area of microeconomics, where so far only rudimentary analysis has been carried out.

12.3.6 Convergence

The convergence behaviour is similar to that reported by Marchuk [Mar82]. He finds that the *method of minimum residuals*[3] has a peculiarity that is important in practice: namely, that the initial iterations converge much more rapidly than the asymptotic rate of convergence. This is precisely what is found for all the variants of the *bargain method.* He goes on to suggest that, to accelerate convergence, it is worthwhile making *occasional* use of a single iteration of a two-step minimum-residual method. This is reminiscent of the acceleration found using the *variable frequency technique.* Choosing the parameters for this extra iteration involves solving a local extremal problem, also much like the maximum profit criterion.

It is found, roughly speaking, that to *halve* the maximum residual in the array, *ten times* the number of iterations are needed.

The *profit method* uses less iterations whereas the *smoothing method* uses less operations. However, the *smoothing method* needs to store the past 6 values whereas the *profit method* needs no extra storage, merely doing the extra calculations the algorithm requires. If the *profit criterion* is applied more often than is really necessary, one finds that this leads to a large increase in the extra number of operations.

In view of the ranking of the convergence at the $1\mu A$ criterion, it is generally found that the Bargain + Smooth method is equivalent to the average of the optimal Game Theory profit criterion method, in terms of the number of iterations. Over a range of convergence criteria from $1\mu A$ to $1000\mu A$ the rank order of the methods investigated is

many iterations	pure bargain
	game 1×5
$\Updownarrow$	bargain + 6-point smoothing
	game 1×5 + optimal exchange rates
few iterations	bargain + 6-point smooth + optimal lookahead

[3]on which the Russians in particular have done a lot of work

12.3.7 Boolean Transform Representation

It is interesting to note that the French researchers, like Robert [Rob87] and many others [DGT85] generally seem to have a better understanding of the convergence issues in finite automata, such as those studied here. They transform the description of the circuit to a boolean space and study its behaviour there, to draw conclusions about its convergence. This is a lot like studying differential equations of fluid mechanics in phase space where their long-term behaviour is much clearer than in the time domain. This approach seems to overcome the limitations of the analytical A-stable methods [Mir81].

12.3.8 Curious Behaviour of Profit $\times 2$ Case

In the study of the *Game Theory* simulation methods it was reported that if the *profit criterion* is applied at every iteration or at every second iteration the system fails to converge at the 1μA criterion. More patient simulation reveals that while this is certainly true, as soon as the $\times 1$ or $\times 2$ application ceases, the array starts to converge again.

It is found that the face voltages and currents remain practically steady for a very long time[4] - and if stopped, say, at 1000 iterations, then the system suddenly 'breaks' out of this mode and converges, but takes 2697 iterations. Strangely, the $\times 1$ case 'sticks' at 68999 μA whereas the $\times 2$ case 'sticks' at the much higher 375085 μA residual.

Closer study of the fine structure of the simulation process, using a 53-bit mantissa, reveals that in fact both the face voltages and the face currents are very gradually changing by a small amount all the time - the voltages, without any oscillations, the currents continually oscillating with a zig-zag form. Depicted in figure 12.2 on page 216 are the voltage, current and maximum residual around the 'break' point, from 970-1070 iterations.

[4]as long as the $\times 1$ or $\times 2$ profit criterion is applied

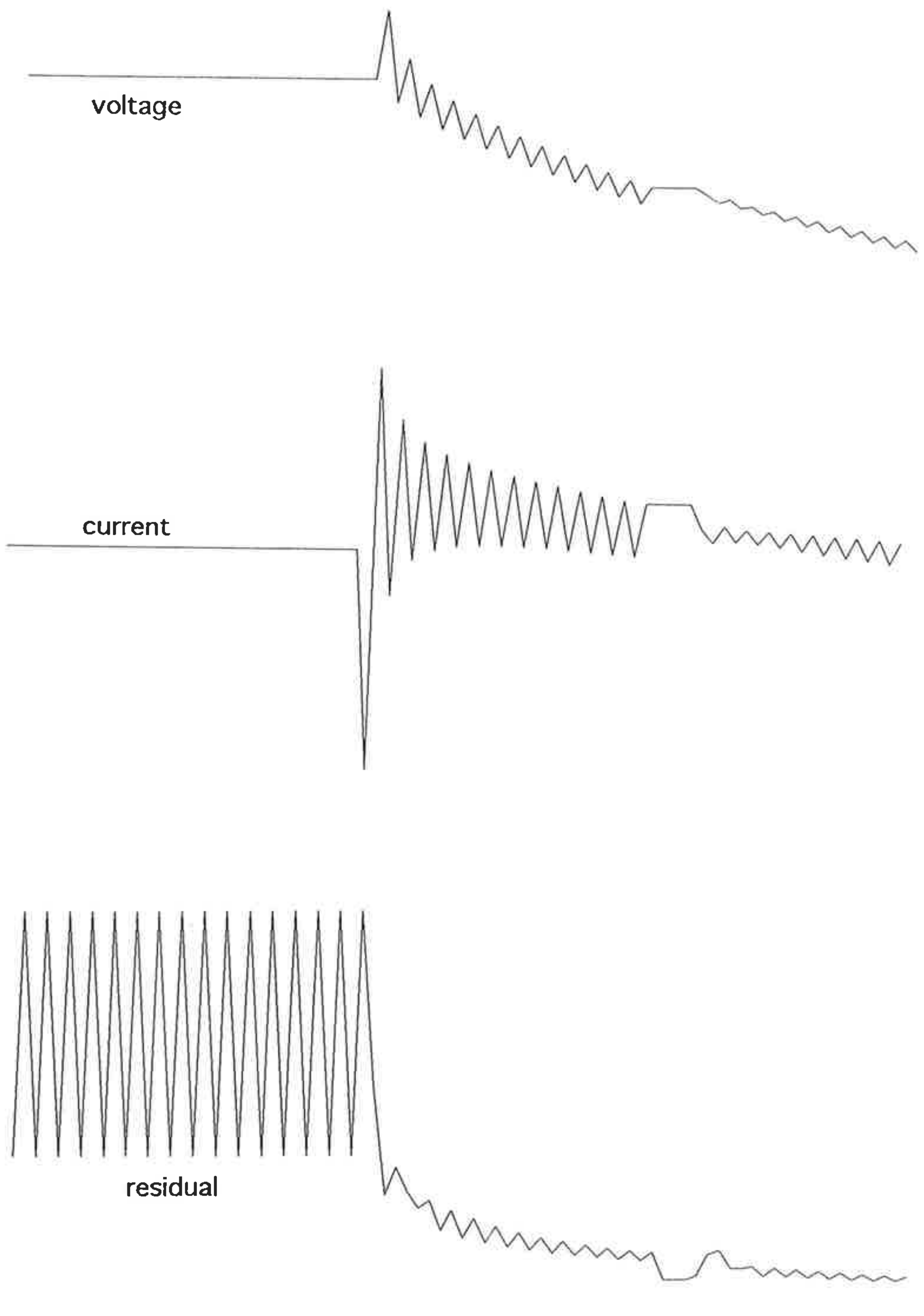


Figure 12.2: 'Break Point' Behaviour in Game Theory ×2 Case

12.3.9 Annealing/Relaxation Similarity

Experiments show that some mixed frequencies of application of the profit criterion give faster convergence than the pure frequency gives. It seems that the profit criterion acts either: as the economic analogue of simulated annealing [Rut89], with extra and unexpected applications jolting the system out of some local rut; or as an aggregation method [CSV88] whereby a slowly-converging relaxation algorithm is interrupted every Nth step, and restarted from another point, because the points toward the end of sequence are confined to a part of the iteration matrix corresponding to eigen-values of magnitude close to one.

12.3.10 Chaotic Behaviour

Chaotic behaviour at a very fine level is produced by the game theory method, as seen in many previous plots in this thesis, and it has one of three explanations.

When the operation of a device is described by differential equations, then an idealized model replaces the actual device. Since every mathematical idealisation involves the neglect of small quantities, then the question arises of how much distortion of the original phenomenon has been subtley introduced. Mishchenko and Rozov [MR80] have studied this problem in detail, and report that, over a long time interval, the differential equation solution can show periodic oscillations with constant or only slowly-decreasing amplitude, which are not in fact present in the original phenomenon.

The situation with spurious oscillations is even worse when, not only are small quantities ignored when the limit is taken in the process of deriving the differential equations, but especially in the next step when they are being solved by finite difference methods. The chaotic behaviour occurs in the modelling by nonlinear, time-dependent differential equations or a system of difference equations. It is often the solution of the equations exhibiting the chaotic behaviour, not the physical system being modelled. There is the danger that a particular numerical method employed to obtain a solution produces the chaos. Tizwell *et al* [TMVK92] note that to avoid this kind of *contrived chaos* whilst retaining accuracy and stability it may be necessary to avoid explicit numerical methods and turn to implicit ones. In solving systems of nonlinear differetial equations this leads to finding the solution by using, e.g., the Newton-Raphson iterative method.

This problem has been subject to sustained attack by Tizwell *et al.* The first order explicit Euler method for solving the equation $\frac{dw}{dt} = f(w)$ is in the form of a sequential recurrence relation $w^{n+1} = w^n + \lambda f(w^n)$ with $\lambda < \frac{2}{\mu}$, where μ is the maximum value of the real part of the eigenvalues of the associated Jacobian matrix $\partial f/\partial w$ at any time t_n. Tizwell employs the alternative form $w^{n+1} = w^n + \lambda f(w^n, w^{n+1})$, the major benefit of which is that it is often possible to obtain the solution explicitly even though the method is technically implicit. He provides many examples showing that this implicit-used-explicitly numerical method has superior convergence properties to the Euler method, and can be used with a much larger time-step. This has already been mentioned in the case of timing analysis in the review by Newton and Sangiovanni-Vincentelli [NSV84].

The third possibility is that these oscillations are the manifestation of exchanged values being reflected from the array boundaries and producing an interference pattern, but much more systematic investigation would be needed to show that.

12.4 How do these Methods compare?

12.4.1 - with direct methods

In the direct methods where differential equations describe the circuit, differentials are approximated by finite difference equations of the node voltages at each time step. This is expressed as a predominantly diagonal matrix equation, which can be solved in various ways. In relaxation methods, the *successive* liquidation of residuals is performed, since this is natural on a serial machine.

All other methods concentrate on nodes, and use devices as auxiliary; in this model it is the opposite: concentration on devices, with 4 nodes per device as auxiliary quantities. Concentrate on nodes leads to integration with respect to time and to relaxation techniques, whereas concentration on devices leads to the present contact simulator. It amounts to the difference between solving the equations $\frac{dx}{dt} = f(x;t)$ globally over the whole time interval, and matching currents locally using $\Delta i = 0$ through each processing-element face by applying Kirchoff's Law at each time step, using what might technically be known as *simultaneous* liquidation of residuals. Another way of looking at it is that, instead of solving a matrix equation on a serial machine, it uses a matrix machine to solve a serial problem.

12.4.2 - with serial methods

It was found, on a Sun SPARC-4 computer, that SPICE2G takes 28 CPU seconds[5] per Monte Carlo simulation for the adder cell previously described; this means that to get a good fit to the distribution needs 500 runs taking a bit over 4 CPU hours[6]. So these programs are best run overnight in any case. However, here there is only simulations for the smaller flipflop circuit, without a full circuit model installed, so no direct comparison can be made.

12.4.3 - with parallel methods

The SIMD mesh-connected computer system is a simple and cheap method. Any device model whatever can easily be put into the processing elements. Herr and Barnes [HB86] found that this ability to modify the model was essential when modelling real processes. Other methods depend a lot on the nature of the mathematical model and how it is implemented, whereas

[5]with other users, 70 seconds elapsed time

[6]with other users, just under 10 hours elapsed time

here one does not have to worry about the mechanism for solving the equations - there are no equations, in the traditional sense. In particular, since there is also no partitioning in the usual sense, problems arising using other methods are irrelevant: there is no chance of putting nodes on the wrong subcircuit and no chance of choosing a wrong order of evaluation of subcircuits. In addition, the array has a better convergence criterion than many other methods. Like the relaxation approach, this method deals with the data. Since only Kirchoff's laws need satisfying then the closeness of the nodes to the correct solution at any time is known absolutely and no other error estimation is needed. This means that if the system converges then it unquestionably converges to the correct voltage node values.

Set against these points are the disadvantages: that it takes a long time to converge if high accuracy is desired; it does not take advantage of circuit latency; and the convergence behaviour has not been properly explored.

12.5 What was Overlooked?

Issues overlooked or not yet addressed but still of importance include:

circuits and models Actual MOSFET models need to be installed - suitable ones are noted in the next chapter. The modelling of floating capacitors and the effectiveness of the game theory algorithm on them has not yet been addressed.

adaptive algorithms All the results were improved, sometimes greatly, when optimal values were used. These were found only after the event, by extensive systematic experimentation. No attempt was made to complicate things at this early stage by adding an implementation of an adaptive algorithm for these. In particular, these should be examined: a different lookahead value for each face of each element; an adaptive smoothing of the profit criterion values; a different exchange rate for every element; and, most difficult of all, perhaps, a different game theory frequency of application for each element.

profit criterion It is not known whether the order of choosing the maximum profit value makes any difference. At present, the program searches each of the 16 possible combinations of node voltages on the 4 faces and uses the last maximum one it finds - even though there might be more than one combination with the same maximum.

12.6 Conclusions and Recommendations

The general questions is: Which is better? the greater cost of a tightly-specified fabrication process, or the lesser cost of a loosely-specified process? and have found that a full stochastic approach is needed to answer it. Theoretical methods of yield estimation need general purpose computers and so take time from workstations, so off-loading circuit simulation onto special computing engines makes sense. Such special architectures are in any case attracting a lot of attention in recent years, for all phases of the VLSI design process.

It is recommend that lots of work take place on automatic adjustment of weights and lookahead, and on convergence conditions for the contact method, in particular the boolean methods of Roberts and others.

For the lookahead random study, an adaptive algorithm, if it can be found, would be preferable - research is wide open for this.

Designer workstations seem too slow for any of the Monte-Carlo methods, so practical implementation in these systems requires an alternative approach to sequential machine computation. Methods using hardware designs with SIMD parallel processing systems such as studied in this thesis might make these workstations the preferred simulators, although much work is needed to determine if feasible algorithms can be found to migrate existing VLSI design tools from serial and MIMD implementations to SIMD architectures.

Chapter 13

Implementation

Here I suggest implementations of both a methodology for obtaining timing yield estimates, and of laying out the array modeller in hardware.

13.1 Heuristic Timing Estimation

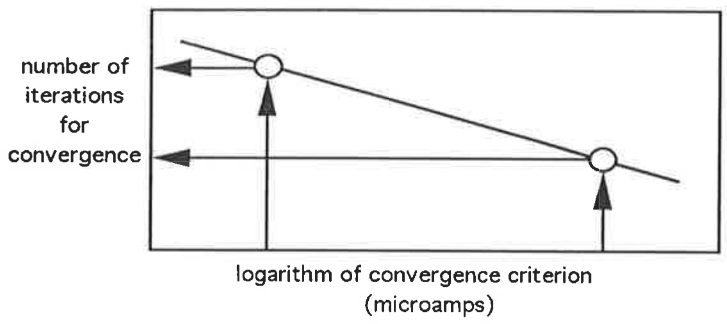
As part of the normal design cycle stochastic tools need to be run early and regularly, because of the invaluable feedback they afford. Given the nature of a circuit and the accuracy of forecast required it is a simple matter to estimate the time needed to get a good estimate of the CDF[1] as discussed below.

Even though the convergence rate of this simulator is somewhat slow for high accuracy work, it is characteristically very much faster for lower accuracy simulation. This suggests a simple methodology where accuracy is traded for simulation time. Since very high accuracy is only required for the final run, then a good idea of how the system design is proceeding can be got using relatively low accuracy estimates, which run a lot faster and thus can be run regularly without penalty. In addition, it is possible to choose to run the *profit* method at the lower accuracies, leaving the *smooth* method for higher accuracy runs overnight.

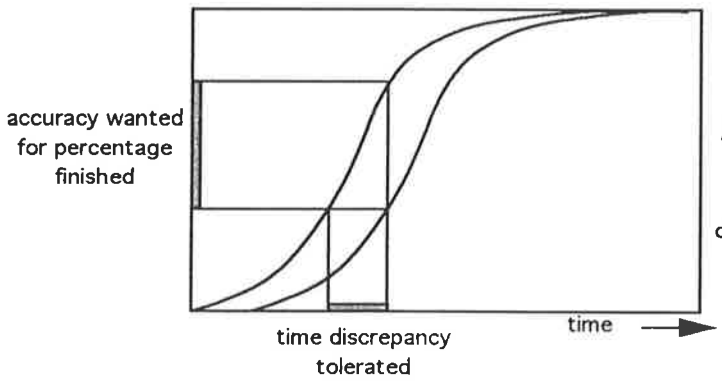
[1]Cumulative Distribution Function

Imagine that the workstation is equipped with the computing engine capabilities incorporating the *smooth* and *profit* models already investigated, and regular checks on the progress of the stochastic aspects of the design need to be made. If inhibited by the unknown and perhaps long time this will take, this method will help choose a rough accuracy $\leftrightarrow$ time tradeoff.

Suppose that the system has already been calibrated early in the design phase (perhaps overnight) at one high and one low accuracy point, in terms of the maximum current residual convergence criterion. Because it is known that the log/log curve is a straight line, already found in figure 10.25 on page 160, it can be calibrated for this particular circuit from just these two points, as illustrated below.


number of iterations for convergence
logarithm of convergence criterion (microamps)

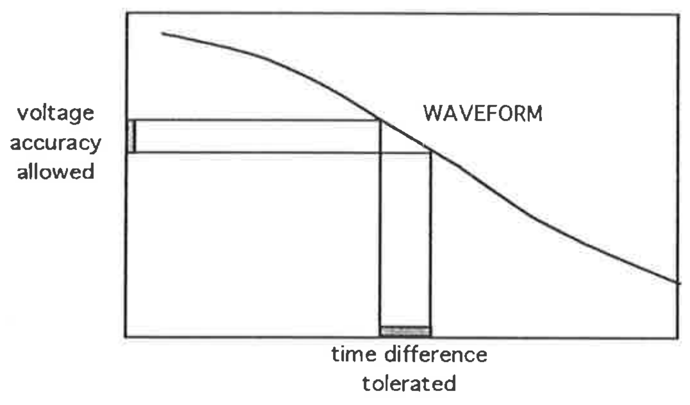
From the initial approximate CDF[2], choose the proposed percentage accuracy from the diagram below.


accuracy wanted for percentage finished
% done
time
time discrepancy tolerated

Simulate the circuit with nominal parameters to obtain the circuit wave-

[2]cumulative distribution function

form near the design operating point, and use this time difference to estimate the corresponding voltage difference as shown below.



This voltage tolerance translates to a current tolerance at the same point in the circuit where the voltage is measured. Obtain this by simulation of the circuit, using the same engine. The number of iterations needed is then read from the calibrated graph already obtained on the previous page, and combined with the characteristics of the engine (number of bits, etc) to give an approximate simulation time in hours and minutes for that simulation. Repeat this procedure, choosing a lower CDF accuracy, if it is decided that can only afford less time for this stochastic estimate.

All this assumes, of course, that a simulation engine of the kind assessed herein is available and attached to the workstation, as a normal adjunct. Whether this will ever be the case depends on the cost of such an item, amongst many other things, and in particular on whether such an engine can run all the other design tools normal to the VLSI environment. i.e., rule checkers, placement and routing, behavioural and fault simulators, timing verifiers, logic verifiers, and so on. Further research is needed to assess whether suitable algorithms exist or could be adapted to run successfully on this engine. This is an unaddressed area for SIMD, although an immense amount of work has been done in this area for the MIMD architectures.

13.2 Implementation Considerations

13.2.1 Accurate Circuit Models

Weng, Yang and Chern [WYC87] find it essential to simulate circuits with a model that guarantees charge conservation. They obtain an accurate threshold voltage model intrinsic buried-channel MOSFET operation in the subthreshold, linear, and saturation regions, which is also valid for the short-channel device. The charge model equations can be used to model the intrinsic capacitances. The current characteristics calculated from the model equation are found to be in good agreement with experimental results.

A comprehensive examination in depth of new small-geometry MOSFET models is given by Ferry, Akers and Greeneich [FAG88].

13.2.2 Algorithms

The choice of the contact algorithm for simplicity in hardware implementation has already been seen when Lewis [Lew88] chooses the forward Euler integration algorithm, largely rejected for software simulators, and discovers that it leads to a fast, but simple and inexpensive, hardware accelerator.

Gallivan *et al* [GJMW91] warn that a code tuned for large grain, vector multiprocessors might be poorly suited to a massively parallel, SIMD machine.

Parkinson and Liddell [PL83] also echo this, concluding that the best algorithm developed for a serial machine is not usually suitable for parallel computation.

13.2.3 Partitioning

There is a need to consider partitioning of algorithms to maintain their numerical stability, and to minimise execution time. Eager, Zahorjan and Lazowski have some useful observations to make on how speedup affects regularity of algorithm in VLSI systems. Deutsch [DLS86], when creating powerSPICE, found that the best partitioning for design, for example, was not necessarily best for simulation. This means, for example, that forming the Thevanin equivalent circuit might not be the best partitioning, either.

McCreary and Gill [MG89] present an interesting theoretical way of partitioning tasks onto parallel machine PEs, which determines the grain size[3] as it does so. This might prove useful. Their method aggregates grains into 'clans' and computes the 'costs' of various strategies. The algorithm inputs a labelled dataflow data-dependency graph of the program, and outputs the grains to be run on the parallel processors.

13.2.4 Models of Parallel Computation

Skillicorn [Ski91] believes that a major reason for the lack of practical use of parallel computers has been the absence of a suitable model of parallel computation, because many existing models are either theoretical or are tied to a particular architecture. He asks *'How can an appropriate data type be constructed whose manipulations are amenable to parallel execution?'*. Dataflow is relatively architecture independent, has great descriptive simplicity, and can exploit all parallelism present in a computation; the PRAM model is the most popular theoretical model for parallel computation, and many of the algorithms developed for the model are synchronous and SIMD in character; but Skillicorn concludes that the most comprehensive data-parallel model derives from the Bird-Meertens formalism because it can handle a wide variety of types. Prins [Pri90] has also done work in this area.

13.2.5 Architecture

It seems that the present design is an embryonic massively parallel computer. Gabriel [Gab86] describes massively parallel computers as implementing a fine-grained parallelism, in which small processors running small or identical operations communicate frequently. He looks at the NON-VON computer

[3]defined as a set of program steps executed sequentially by a single PE

with its two categories of processors, the small processing element and the large processing element. The former operate in SIMD mode under the control of the latter, and it shares the SIMD programming style of the Connection Machine.

Arden and Ginosar [AG82] consider that a multiprocessor having a large number of processors would be very general, but large systems of this type are not feasible, because of storage contention, the complexity of complete processor-memory block interconnections, and the efficiency loss of cache memories[4]. On the technological side, VLSI circuits are not well suited to the implementation of complete interconnections between many processors and memories. These are some reasons why there has been significant interest in loosely coupled multicomputer systems. The limitations of the number of processors and the interconnection complexity are eliminated in such systems. However, the overhead due to multistep message passing when the interprocessor communication is frequent, is substantial.

To help overcome some of these problems, Charlesworth [CG86] urges implementing systems with only the minimum amount of internodal communication required to solve the dominant problem of interest. He notes that the size of the mesh can be extended indefinitely, but doing so also indefinitely extends the distance between arbitrary nodes. Meshes are best for algorithms where the data can flow locally, step by step across the system. To improve the resolution of the model, the number of node elements can be increased, or a finer time step used, but either method drastically increases the computation time.

Parkinson and Liddell [PL83] raise and discuss some very good questions concerning distributed computers, namely: *What class of problems are highly suitable for a given multiple processor system? What class of problems are high unsuitable for a given multiple processor system? What type of performance is it reasonable to expect from a given multiple processor system?*, and go on to discuss the ICL DAP, a 4096-element SIMD processor array embedded within a store

[4]however, these might all be overcome by better, i.e., new, algorithms

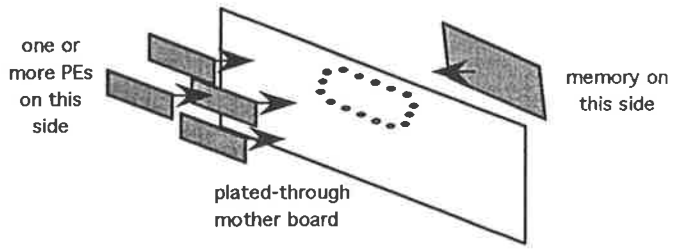
module of a conventional host computer.

13.2.6 Array Processor

Renterghem [Ren89] explains that in a transputer system, there is no bandwidth saturation as the system size increases, no capacitive load penalty as more transputers area added and no communication bus contention. If a processor communicated only with its direct neighbours, the ratio of communication to computation remains constant if we use more processors. Scaled speed-up is not limited by Amdahl's law. Duncan [Dun90] and Merrow and Henson [MH89] provide a survey of designs and architecture.

13.2.7 Physical Layout

The very mature technology for creating dense memory chips is unsuited to creating microprocessors, and thus a custom CPU chip, separate from a standard dense memory chip, but with compatible pin-outs, attached to each side of the one board, as shown below, might be feasible.



Mazumder [Maz92] investigates layout optimisation for processor-array networks, using a new layout style based on polyominoes. If any appropriate shape geometry is selected for the processors, a specific interconnection network can be area-efficiently mapped on a VLSI/WSI[5] chip to maximise the chip yield and operational reliability. He finds that the square mesh with redundant processors provides high yield and operational reliability.

[5]wafer-scale integration

Esonu *et al* [EAKHAK92] also examine the effect of objective function optimisation on the architecture of systolic arrays. In each case, when the objective function is optimised, a different systolic array is produced. They use purely architectural arguments, which rely on the assumption that the maximum clock frequently is independent of the processing element count, recognising that this is not necessarily true in a monolithically integrated circuit. Algorithms are mapped into a systolic array using a parameter dependency technique. Features that can be optimised are the: fault-tolerance, propagation delay, throughput, silicon area, routing complexity, speedup of computation for the systolic array, or some combination of these. They find that to minimise the delay required for the data to propagate through the VLSI systolic architecture it is desirable that routing between cells be to nearest neighbours.

13.2.8 Scalability

This feature means that extra PEs can be added without fundamental algorithm alteration. To be scalable implies, not broadcast, but an instruction-systolic array of the type introduced by Lang [Lan86] and used by Schroder [Sch89]. It consists of a mesh-connected array of identical PEs. Each processor has a small local memory and a register readable by its N,E,W and S neighbours, through which instructions are pumped from N and W, like the illustration in figure 13.1 on page 230. They claim that it combines the advantages of systolic arrays with the idea of a universal machine.

13.2.9 Technologies

Garcia and Sriram [GS82] consider that, of the MOS technologies, nMOS combines a good speed/power product, high packing densities and low fabrication costs.

Mulitinovic [MF86b], on the other hand consider the use of Gallium Arsenide technology in the implementation of high performance processors, and

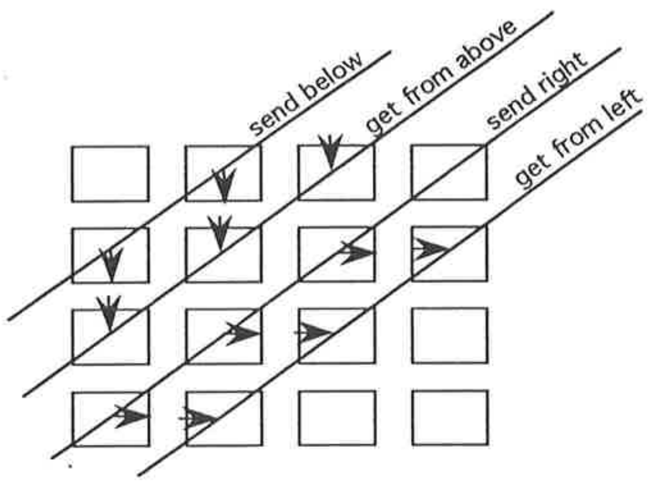


Figure 13.1: Illustrating Instruction Pipeline for Scalability

finds that GaAs is inferior to silicon in cost and transistor count capability. It is not sufficient merely to copy existing silicon designs into GaAs. Characterised by a low transistor count, a high ratio of off-chip memory access delay to on-chip datapath delay, low gate fan-in and fan-out, and low yield, a premium is placed on simple designs in the GaAs environment, and every transistor must be justified.

13.2.10 Wafer Scale

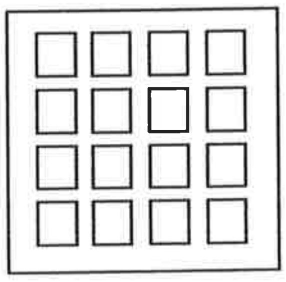
McMinn [McM82] gives a practical example where slightly more than doubling the area causes a near 6 fold decrease in yield, while discussing the importance of the tradeoff between die size and projected yield when partitioning the system for a custom chip, and Peltzer [Pel83] stresses that WSI can improve system reliability and reduce the yield loss caused by small random defects by the use of redundant circuits.

Recently, Singh and Youn [SY90] have presented a scheme that can reconfigure rectangular arrays to avoid defects. They claim that the yield is much higher than other methods with the same degree of redundancy.

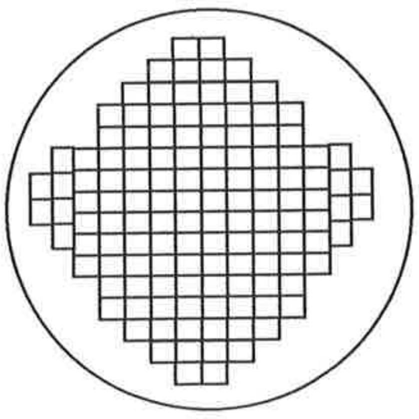
13.3 Cost

The best solution depends on the amount of use to be made of the model, and whether larger assemblies of transistors will need to be simulated. A good discussion of the fundamentals of costs is given by Muroga [Mur82] for LSI and VLSI, with references for more detailed results, although the actual examples are dated. It must also be borne in mind that computer software costs of running Monte-Carlo simulations are quite high. Considering that a typical design may contain half a dozen cells that lie on critical time paths this means that, after just one redesign of each, there is a large cost blowout even before fabrication. This cost is not immediately obvious and is often overlooked by those who have 'free' use of large research computers. However, the necessary costs need to be seen against the possibility of the failure of a single fabrication run.

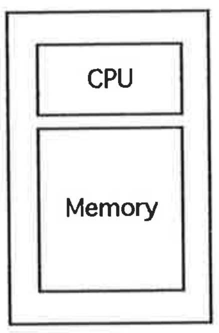
There are many different ways of arranging a system like the array simulator in practice. Speeed may be traded for size to get high yield and hence low cost. The *PEs* may be packaged so that a number of them fit onto one chip; or if small, somewhat more of them might fit onto one wafer, which is configured for redundancy and fault-tolerance; or if the PE contains its own memory and is thus rather larger, it could be in one chip by itself. These various options are illustrated below.



one chip - 16 PEs

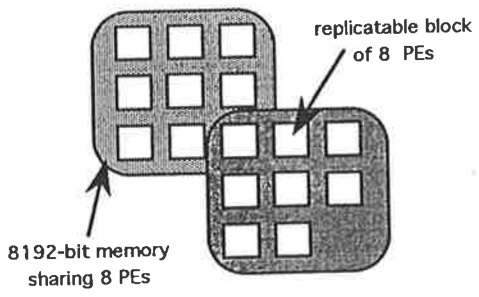


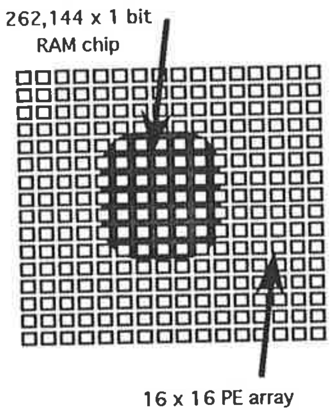
1 wafer - 128 PEs nett



single-chip PE

Pursuing the idea, mentioned earlier, of having a *PE* on one side of a board and a memory on the other: the memory could be shared amongst many *PEs* since each one only needs the 20 or so model parameters and the neighbour voltage and current – no more than 32 words. For example[6], an 8K-bit memory could contain 256 32-bit words (real numbers), and so be shared by 8 *PEs*, using the above assumptions; or at the other extreme one 256K-bit memory, while being more expensive than the smaller ones, could share a 16×16 array of 256 *PEs*, both options illustrated below. Perhaps one





of the 64K-bit multiport video RAM chips might be adapted to this sharing, for speed purposes, but to offset this is the bit-stream nature of the chip, and the need to latch the address in two stages, which slows it up; this is in addition to rather complex interface circuitry between the *PE* array and the memory chip. All these options would need further exploration to find the optimum layout, with the yield/area formula being crucial to finding the best arrangement as far as minimum cost is concerned.

Even the transputer array might eventually be practical, if it has enough local memory, with the possible advantage that it could be used in MIMD mode as well as SIMD mode, as necessary. One transputer costs under \$200 now, so a board with 10×10 of them would cost under \$20,000, plus support chips, although in the terms of this thesis this is not considered cheap.

[6]without using current costs, since they date so quickly

Chapter 14

Future Research Directions

Here I look at the current state of parallel computing and try to assess where the results of my research might fit in.

14.1 Interest in SIMD and MIMD

There is presently tremendous research interest in both massively-parallel systems and distributed systems. The former are generally of the SIMD architecture and the latter MIMD. An illustration of the broad classification for SIMD and MIMD architectures, following the schemes of Lopez [LV90], Christ [CT84] and Kung [Kun87], is shown overleaf.

Small parallel computers implement a coarse-grained parallelism, in which relatively large processors running relatively large, mostly independent computations communicate infrequently. Massively parallel computers implement a fine-grained parallelism, in which small processors running small or identical operations communicate frequently. These cover the full range of pipelined processors, systolic arrays, neural networks, multiprocessors, vector processors and processor arrays. Skillicorn [Ski88], amongst others, offers a comprehensive classification of architectures.

Over the past decade, Swartzlander and Gilbert [SG82] conclude that the distributed processor approach appears most desirable for supersystems, but

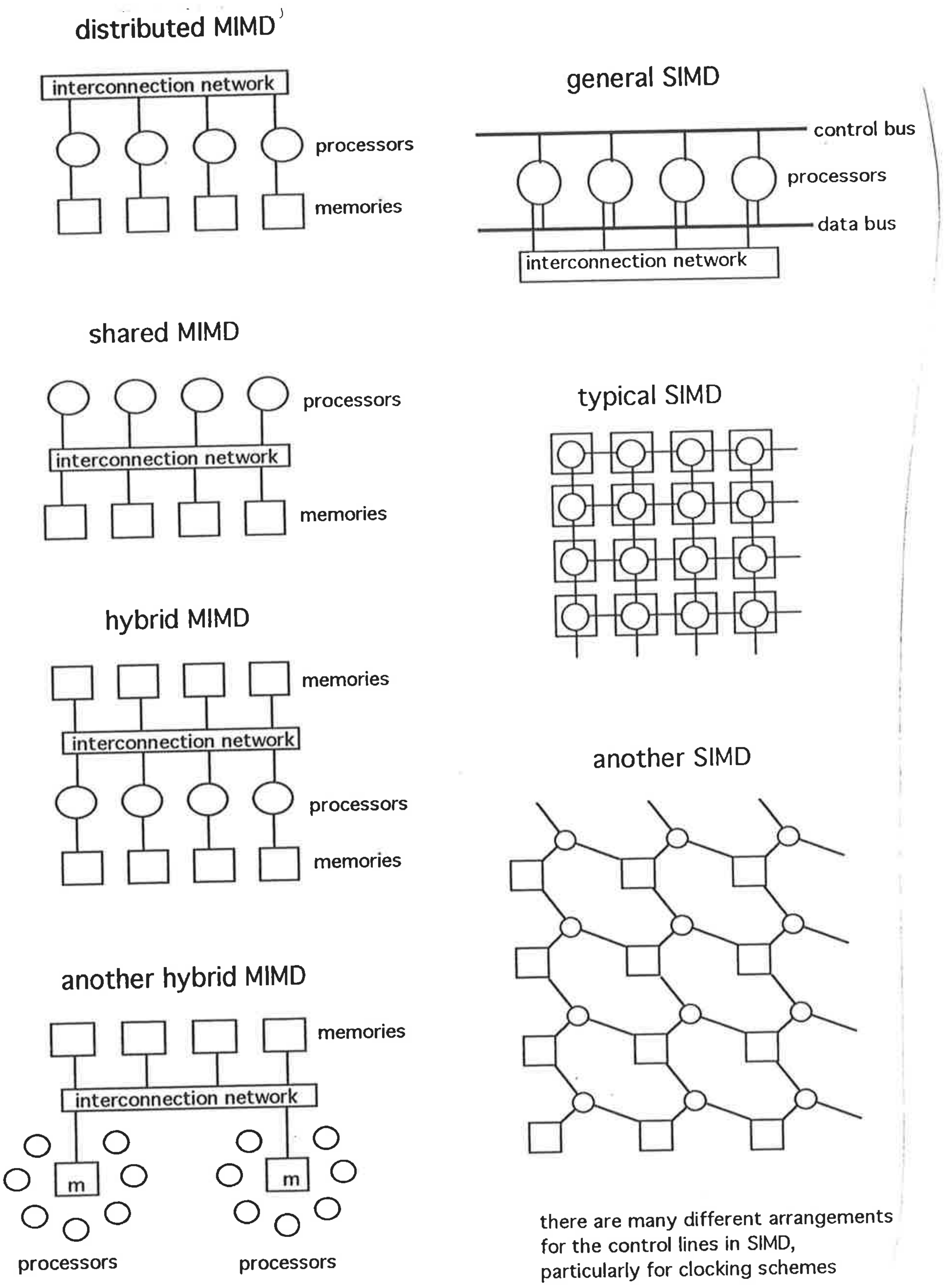


Figure 14.1: ILLUSTRATING MIMD AND SIMD ARCHITECTURES

will required improved interconnection networks.

Gabriel [Gab86] finds that parallel and pipeline processors lack the required flexibility for many supersystems applications, so attention has been focused on distributed networks. The most efficient algorithms had a common design theme in that all are multiple instruction stream, multiple data stream (MIMD) devices. He recognises that the major problem to solve in building a massively parallel computer is how to interconnect a very large number of processors and memory modules, and concludes that for more than 10,000 *PEs*, the cost of the crossbar switch is prohibitive and its size unmanageable. Amongst Massively Parallel Machines, the Connection Machine uses the hypercube connection scheme, and the NON-VON can support SIMD, MIMD (multiple-instruction multiple-data), and MSIMD (multiple SIMD) operations.

Gottlieb [GGK+83] rejects SIMD machines in favour of the MIMD paracomputer model, which his simulation studies show to be effective for both fluid-type and particle tracking calculations. Most recently-introduced multiprocessors have a few dozen processors connected to a shared memory over a common high-speed bus. He finds that many problems hitherto considered unparallelizable have, in fact, a substantial content of exploitable parallelism, and that the speedup is reasonable for up to 16-20 processors, but little is gained by increasing the number beyond that.

Requa and McGraw [RM83] investigate the architecture of the Piecewise Data Flow computer (PDF), a heterogeneous multiprocessor proposed for very high performance computing, which they claim blends the strengths found in SIMD, MIMD, and data flow architectures. The SIMD machines are very cost effective on vector processing. Almost all currently proposed data flow architectures have a large pool of homogeneous processors. Instructions are sent to processors as soon as all data dependencies have been satisfied. The PDF architecture has heterogeneous processors (memory, scalar, and SIMD) and instructions are sent based on data dependencies. For long-term high-performance software, they believe the advantage must favour the data flow-like approach.

Charlesworth and Gustafson [CG86] find that many problems have sufficient potential parallelism to utilise 10,000 to 100,000 concurrent computing nodes, and recommend implementing systems with only the minimum amount of internodal communication required to solve the dominant problem of interest.

Dongarra [Don88] looks at experimental architectures while Lopez and Valimohamed [LV90] deal with hybrid systems having properties of both shared-memory and message-passing systems.

Both SIMD and MIMD algorithms, for the same problem, are pursued, particularly for the hypercube architecture [RS90], and for the connection machine architecture [She91], for matrix and signal processing algorithms. A vast and comprehensive description and classification of the array processor types and algorithms exists by Kung [Kun87].

The overview by Vorst [VD90] confirms that the interest in parallel research is driven by the availability of large machines and the need for efficient parallel algorithms to run on them. He reports much recent interest in systolic array algorithms, message-passing systems, shared-memory systems and vector supercomputers. Tamura [Tam91] suggests that the difference between general and specific purpose parallel processing systems is not very clear, since any specific purpose system may be used for other purposes whereas a general purpose system has limitation for some specific use; thus the differentiation is rather arbitrary. This is in the context of examining the Cellular Array Processor (CAP), a SIMD machine. The CAP has 4,096 *PEs*, each of which processes 1 or 8 bit data. Smitley and Iobst [SI91] propose that the SIMD concept be also viewed as a model of computation.

Maresca and Fountain [MF91] edit a recent overview of massively parallel architectural research. Originally developed, and still much used, for image and pattern processing, these have been given a stimulus to extend application to new fields, by modern VLSI design capabilities. Specifically covered are the AIS, the DAP, the MasPar and the CM-2. Li and Stout

[LS91] examine architectures designed to take advantage of reconfiguration of the *PEs* to speed up massively parallel processing, whereas the opposite view is pushed by Distante *et al* [DSSSG91], who believe that fault-tolerant wafer-scale methodologies will be more successful. There is also a lot on the relation between parallel architectures and neural networks [cai90].

Skillicorn [Ski91] believes that, rather than developing more specific architectures and algorithms, much more attention should be directed towards creating suitable models of parallel computation, urging the Bird-Meertens formalism as a coherent approach to parallel programming.

14.1.1 Summary

Originally, speedup was sought on vector machines, (CRAY, STAR, CDC 6400, etc.) to which few have access. Then speedup was sought in algorithms on sequential machines, to which many had access. Eventually distributed processing became available to workstations in MIMD form, and the original SIMD was lost sight of. But SIMD is much more than just the original vector architecture. Modern SIMD involves the Massively Parallel Processor architecture.

It appears that very regular algorithms have been mapped to SIMD architectures without much flexibility, while simulation and VLSI design has been mapped to more general-purpose MIMD architectures. This has led to the perception that simulation cannot be successfully carried out on SIMD machines.

The existence of simultaneous research into all of these areas stresses the tremendous disagreement that exists about a clear way forward.

14.2 High Performance Computers

Bell [Bel89] in a very widely-circulated paper, predicts that *"A vast array of new highly parallel machines are opening up new opportunities for new applications and new ways of computing"* . He then reviews SISD, SIMD, MIMD (shared) multiprocessors and MIMD (message-passing) multicomputers of many kinds, including engineering workstations. Specifically mentioned and described are the computer systems amongst those listed in the table on this page, and many others, along with some very high performance computers specialised for one particular problem.

This is a very impressive range of systems. By any reckoning some of these are great technical achievements indeed, and many of these companies seem to done all the right things, in design, production and marketing.

Computer System	Company/Producer	Type	Type
WARP	G.E.		MIMD
CM-2	Thinking Machines;	SIMD	
MultiFlow	Fisher/MultiFlow Corp.	VLIW	VLIW
DAP	ICL /Active Mem. Tech.	SIMD	
AIS-5000	Applied Intelligent Sys.	SIMD	
FPS-array	Floating-Point Sys.	SIMD	
CEDAR	Uni. Illinois		MIMD
iPSC80	Intel		MIMD
Monarch	B.B.N.		MIMD
UltraMax	Encore		
X-MP & YMP	CRAY		MIMD
RP3	I.B.M.		MIMD
MasPar	MasPar Comp. Corp.	SIMD	
PIXEL Plane	Uni. N. Carolina		
GF11	I.B.M.	SIMD	
MPP/FX	Alliant		MIMD
Graphic Super	Ardent		MIMD
Delta	Touchstone		
Hypercube			MIMD
Kendall Square	Kendall Computer		MIMD
Transputer	InMOS		MIMD
NCUBE	Caltech		MIMD
MultiStage CS-2	Meiko Scientific	SIMD	MIMD

None-the-less, at the time of writing[1], a majority of the companies listed are either:

- in actual receivership, or
- filing for bankruptcy proceedings, or
- supported by massive injections of government money,

[1]from financial press reports, September 1992 – January 1993

and in some cases the products have simply been discontinued, or the companies have merged, i.e., there are very few profits in digital array processing just yet.

14.2.1 Discussion

One of the reasons for believing that the above state of affairs is bound to get even worse is contained in Bell's own article: *"Supercomputing has become an issue of national pride and a symbol of technology leadership"*, which, by all past experience, dooms it to ignominious and drawn-out failure. The seeds were sown in 1984 with the establishment of the NSF Advanced Scientific Computing centre, and culminated with the 1987 publication "A Research & Development Strategy for High Performance Computing[2]", which committed the US government to a large injection of funds in pursuit of a national TeraFlop machine. Many of the companies on the list participate in this scam.

The lack of profitability of the parallel machines above is more revealing than the direction of all the current research put together. One main thrust of the thesis of Maly [Mal90] and others is the need for profitable manufacturability of ICs. Since profit is technically a measure of the incentive to re-invest, vanishing profits means that the product does not figure in the future at all. When money is diverted from productive to unproductive pursuits by force and by fraud, and when profit is no longer a dominant consideration, companies get into financial trouble, and in particular items *not* needed are produced in abundance.

The significance of all these losses is that investment in microelectronics will be withdrawn on a large scale, and this means it will be harder to produce electronic systems. Crucial questions that should now be asked include: Who can afford these systems? and, What can they get from them that they cannot get from their existing systems?

[2]US Office of Science and Technology Policy

At a time when VLSI CMOS 0.8μ circuits can be routinely manufactured with a million transistors on a chip, and people have accepted that MPP will work since Gustafson demonstrated practically linear speedup for engineering problems on the hypercube computer, the final relevant question is: Systems with a few large processors, or with massively-parallel many?

14.3 Consideration of Problems

It seems that the original hypercube warcry *"with hardware cost a secondary consideration"*, which flowed from publically-funded research institutions, has inevitably lead to a contradiction – because in microelectronic business hardware cost is never a secondary consideration. Cost reduction is about to become quite paramount. What developments might facilitate this?

McMinn [McM82] reminds us that the tradeoff between die size and projected yield is an important consideration when partitioning the system for a custom chip, since slightly more than doubling the area causes a near 6 fold decrease in yield, and test time can really mount up. Peltzer [Pel83] reports that WSI can also improve system reliability.

From the early work of McCanny and Whirter [MM83] up to recent work by Singh and Youn [SY90] there has been steady progress in techniques that can be used to increase the yield of wafer-scale circuits, and produce chips which would otherwise give totally uneconomic yields.

The backplane problem has been attacked very recently in the context of the scalability of the hypercube architecture by Ziavras [Zia92] using optical methods, but it is hard to be convinced yet that this is an inexpensive solution.

There are recent investigations into reconfigurability by Li [LS91] using a polymorphic torus scheme to bypass faulty *PEs*, and aimed at a SIMD MPP with over a million *PEs*. There is also the alternative defect-tolerance scheme for mesh arrays of Distante [DSSSG91]. Lea [LJ91] proposes to use associative processors to ensure scalability and also fault-tolerance, in a system which could produce 1 MOPS per dollar, in volume. In addition there is the work of Wilding *et al* [WTHP91] which concentrates on the applications of cellular-automatons to scientific problems: the importance of this is that it connects with the french boolean-mapping approach[3] to investigating the convergence of SIMD arrays. Thus measures exist aimed at reducing costs.

14.4 A New Question

The main problems concern scalability and cost. Combine that with the need to simulate large parts of the circuit at once, and suddenly there is the need for one hundred thousand to one million *PEs*.

In simulation, because a lot of the circuit is relatively quiescent, then it is reasonable to use only a few processing elements (as in the MIMD case). But in future, if simulating elements of DataFlow computers, which can be fairly active all the time, and if can have WSI with very many elements ...

Christ and Terrano [CT84], consider that there are a large number of significant scientific and engineering problems which can be efficiently solved by a array of processors interconnected to a form a multi-dimensional grid, and by taking advantage of powerful commercially available VLSI chips, they design a parallel array of single board computers, containing 256 nodes running in lock step in a SIMD mode with a computational power of 4 billion 22-bit floating point operations per second.

Lopez [LV90] urges the philosophy of solving the largest problems possible in a reasonable amount of time, rather than solving existing problems faster. This certainly fits in with cost-reduction point of view.

With WSI and fault-tolerance redundancy techniques at hand, the original problem that spawned so many MIMD machines: *"How might scarce resources be applied to best effect?"* becomes transformed into *"What is the cheapest way to apply practically unlimited resources to best effect?"*, because with this technology it is possible to have the same number of *PEs* as devices to

[3]previously mentioned [Rob87]

be modelled, and this makes the problem completely different.

Under these assumptions, it is still an open question as to what class of problem it can successfully apply, but it certainly seems that the divide between: few powerful processors, in MIMD; and very many simple processors, in SIMD; is to become more marked in future. Since it is found that around 20 processors is the point of diminishing returns in a MIMD system – with speedup that is difficult to improve on – then this line of development cannot be followed for *cost reduction*, since the only way it can go is towards increasing the power of the 20 processors involved by further division into more sets of 20 processors[4].

It seems to me that the logic of this next phase is inescapable: device physics is reducing to a smaller scale for power; defect density means smaller chips for yield. Thus there is an impetus towards many simple *PEs*, so the interconnection problem becomes intractable[5]. This means that new algorithms are needed which are suited to a 'backplaneless' world. This rules out all MIMD systems except transputer-based systems. It seems that circuit improvement *per se* is now largely irrelevant, and that advances in architecture are dominating research, and in particular that new algorithms are essential to effectively utilize these advanced architectures.

Thus future development might have to concentrate on SIMD MPP machines, where there is more scope for both new engines, physically, and new algorithmic development, theoretically.

Fully switchable MIMD and SIMD is out of the question for more than ten thousand *PEs*, which leaves fixed switches in SIMD or switches that learn to reconfigure, as in neural networks.

[4]although there is much work on the partitioning problem to be done

[5]e.g., Alliant spent millions of dollars trying to produce a 43-layer board and failed

14.5 What about Game Theory?

The simple step of re-interpreting the Newton-Raphson iteration algorithm is crucial to the usefulness of this thesis. It takes the NR technique out of its purely mathematical setting – in the field of differential calculus – and places it firmly in the field of economics, in particular in the game theory area.

Since NR is widely used in fundamental microelectronic mathematics, this places game theory considerations at the heart of electronic modelling for the first time. Since microelectronics has been applied to neural networks in recent years then it is possible that the game theory approach can illuminate neural networks as well. So there appears some very far-reaching possibilities from such a relatively slight change of perspective.

14.5.1 Dynamical Systems & Cellular Automata

After much research into ULSI[6] the opinion of Ferry *et al* ([FAG88] p. 264) is that *". . . cellular automata may be viewed as representations of dynamical systems . . . any computer architecture that aims to maximize the density of active devices, while minimizing the delay inherent in interconnections, leads to a layout endemic to cellular automata. What we seek in future ULSI is not games played with cellular automata . . . but the nature with which cooperative phenomena are exhibited in these systems, and the extent to which these cooperative phenomena offer new techniques for information processing."*

This is a fascinating viewpoint, connecting not only the cooperative and dynamical aspects of the game theory approach but also, *via* cellular automata, the French boolean perspective on convergence of arrays of processing elements.

[6]Ultra Large Scale Integration

14.5.2 A New Paradigm

Thus the incidental reinterpretation, discovered in the course of this thesis, of the NR iteration as a bargaining technique rather than just a purely analytical technique, represents a paradigm shift in IC simulation.

It has been seen above that new algorithms are essential to effectively utilize advanced architectures. It must be accepted that general *inter-PE* communication makes for faster programs – the more information exchange, the better global algorithms work. However, the contact model is based on the hypothesis that only static communication is available, and that algorithms will just have to adapt to this situation. The main point of the experimental work that has been carried out is to explore the question: How much progress can be made under this restriction? and the answer to this so far is that it seems that promising progress can be made, and fast convergence achieved, by inventing appropriate new algorithms. The problem with extensive communication with many *PEs* at once is that more memory is needed to hold values, so more complex and expensive *PEs* are required.

Work that also bears on the behaviour of contact simulators, in addition to the game theory and the boolean/automata fields, is done by Murphy [Mur90] and Shynk and Roy [SR90], all of whom report on the nearest-neighbour convergence properties of perceptrons.

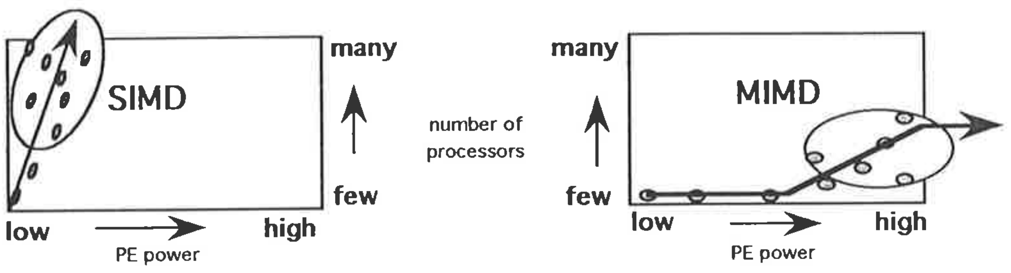
The Game Theory paradigm might be well-adapted for a new generation of wafer-scale SIMD simulation engines.

14.6 Recommendations

It has been seen that cost reduction is mandatory for future MPP systems. The large and powerful MIMD processors can have their own on-chip memory - this has to be counted an advantage for speed - but they therefore have lower yields. The small simpler processors do not have their own on-chip memory but do have very high yields. The sums that could offer some guidance to decision-making in this area have not yet been done.

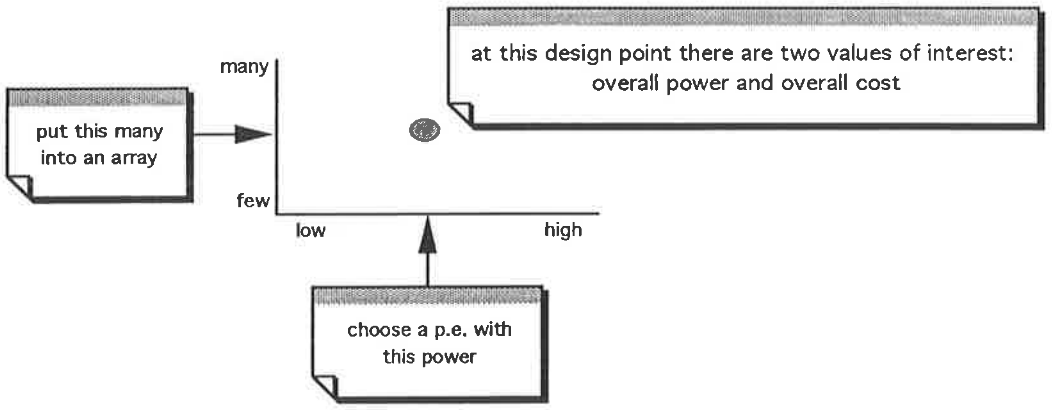
14.6.1 Computing Power at Constant Cost

Production of parallel machines might be summarized in these illustrations, where it suggests a flattening since MIMD speedup falters above 20 pro-



cessors. The development of SIMD and MIMD are rather separate, often because applications run on one kind rather than the other. Now, there are any number of graphs showing computing power as a function of the number of processors, for classes of problems, for both SIMD and MIMD machines (e.g., [Bel89]). But what is really needed, in my opinion, in an era where cost cutting is important, is a depiction of how to move to a computing establishment from an existing one at very similar cost structure.

Consider the situation depicted below. If points are plotted all over this,


at this design point there are two values of interest: overall power and overall cost
put this many into an array
many
few
low
high
choose a p.e. with this power

from actual and proposed systems, it will be found that each graph (power and cost) has a set of level surfaces – i.e., lines of constant power, and particularly, given the power, lines of constant cost – which can give an indication of the available options at a similar cost to the currently owned computing resources. This would be a very useful practical aid to companies for investing resources, as well as for delineating possible future research directions, and its construction is highly recommended.

14.6.2 Iteration & Convergence

Amongst the advantages of the boolean representation of arrays of cellular automata are that one does not have to:

- form circuit equations;
- partition the circuit;
- form the linearized approximation matrix equations;
- or find its eigenvalues.

It is sufficient to work from the local connectivity matrix. Thus it is recommended that the application of the boolean methodology be thoroughly investigated – a good start is Robert [Rob87] and Demongeot [DGT85].

14.6.3 Game Theory

In view of the initial success of the game theory approach, there is an immense amount of work that should be carried out on these related questions:

- what is the extent of the rigour of the analogy?
- under what conditions do economic systems converge, and not converge?
- what is the value of broadcasting price signals to everyone in economic systems, and where does it apply to broadcast SIMD arrays?
- since each Monte Carlo simulation is only a slight perturbation of the nominal case, can the system learn from past runs[7], and how?

[7] *vide* Maly [Mal82]

- can neural network methods be applied so that different criterion strategies that show promise can be reinforced?

Thus it is strongly recommended that a proper theoretical justification of the game theory approach be undertaken, in addition to investigating the application of adaptive techniques to all aspects of the current implementation.

14.6.4 Design Tools

If this thesis is correct in suggesting that SIMD MPP is a fruitful direction in which to proceed for constructing VLSIC simulation tools, then it necessarily also recommends that research be undertaken with the aim of migrating *all* existing design tools from serial and MIMD algorithms to suitable SIMD algorithms.

14.6.5 Summary of Recommendations

As a result of the work in this thesis, the following recommendations are made, concerning:

cost: obtain or construct data from which constant-cost graphs might be constructed for various MIMD and SIMD array systems;

cellular automata: investigate co-operative properties of these systems, and apply boolean methods to convergence studies;

game theory: establish rigor, investigate convergence and develop adaptive strategies;

design tools: encourage algorithmic migration from MIMD to SIMD architectures;

floating capacitors: investigate representation within array for use as finite-element method for interference effects in sub-micron geometries.

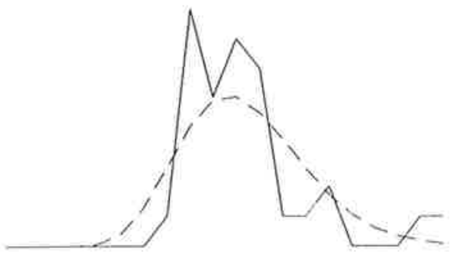
14.7 Conclusion

Because of the great detail in the diversity of approaches to stochastic problems, it has been thought best to summarize the most important conclusions at the end of the respective chapters. There are, however, some points worth emphasizing here.

Some useful discoveries relevant to the application of stochastic techniques to workstation practice have been made. It is good to know the form of the probability density function: *e.g.,* for the mere 33 data collected on page 254, an Erlang distribution has been fitted using just the mean and variance as parameters, giving a sound idea, shown at the bottom of this page, of the progress of the design with respect to its specifications. The results obtained for concatenated structures is quite practical, too.

The Newton-Raphson interpretation as a bargaining scheme has led to the game theory paradigm, which has shown that new algorithms applied to old architectures can be just as efficacious as old algorithms applied to new architectures. Theoretically, there is the application of the whole field of the theory of cellular automata to be explored in this regard, while on the practical side the memory/PE arrangement mentioned previously allows the best of each processor and memory technology to be used.

The future for workstation parallel processing engines looks very bright and appears wide-open to research into new architectures and algorithms, aimed at affordable parallel processor systems with adequate performance, rather than expensive ones with high performance. The various subtle trades-off between cost, speed and size are a fruitful field of examination, and already form a sequel to the work carried out in this thesis. I suspect that the SIMD model, in its wafer-scale-with-redundancy form, combined with new architectures and algorithms, might become the preferred architecture of parallel-processing engines for the workstations of the future.



Appendix A

Multi-Project Chip

This design tests an adder leaf cell which may be used in future MPC designs to construct a simple, fast n-bit multiplier subsystem component, for arbitrary n. The chip design includes two test structures: a leaf cell with four input and ten output points monitored; and a 7 by 4 array of leaf cells, with ancilliary buffers, which produces an 8-bit product from two 4-bit multiplicands. It is implemented using nMOS technology, with λ=2.5 microns.

This report describes our experience in designing a very large scale integration circuit subsystem. It is a proposal for the C.S.I.R.O. Division of Computing Research Australian Multiproject Chip in May 1982 [N-bit Multiplier, Author's Publications]. We compared our calculations of performance with measurements made on the bonded chip to help to establish better estimates from which to predict future design performance in different structures. It is a simultaneous array multiplier using cascaded adders, and has been described in some detail by Lewin [Lew72].

A.1 Formal Algorithm

This is the algorithmic description for an n-bit multiplier block. An n-bit multiplier comprises an array of adder cells, 2n wide by n high. One multiplicand, with bits Y(i), (i=1..n) is fed to the n leftmost cells along the top edge of the array, with y(i) going to cell(row=1,column=i). Y(i) is transferred out of the cell (unaltered) to the cell diagonally down to the right closest to it. The bits of the other multiplicand, X(j), (j=1..n) are fed into the right edge of the array, such that X(j) goes to cell.

The Mead/Conway method of top-down design leading to functional specifications of leaf cells meant also that they (the leaf cells) could be altered at any time but were still guaranteed to fit in with the floorplan. In fact, one large change took place (replacement of red/green function block by multiplexor) at a late stage in the project, and because of the design method this was achieved in two days, each of us concentrating in his own area of expertise.

A.2 Cell Description

Circuit Diagrams

The full adder leaf cell has the following functions to perform, by virtue of the algorithm

(1) to accept X from the right and pass X restored to the left (the restored value is also used within the cell)

(2) to accept Y from above and pass Y restored to the cell diagonally down and to the right, the restored value being used also in the cell

(3) to perform the logical X AND Y producing an intermediate result, XY

(4) to accept the result (sum) from the cell above (Ain) and restore it, producing A and -A for use within the cell

(5) to accept the carry (C) from the cell to the right (Cin) and restore it, producing C and -C for use within the cell

(6) to perform the full adder function on A, B and C and supply the result (Aout) to the cell below and the carry (Cout) to the cell to the left.

These functions are performed by the circuit whose layout is shown in figure A.1 on page 252.

A.3 Timing Information

Adder Leaf Cell

SPICE simulation and hand calculation lead to similar results, but diverge because in the hand calculation we assume depletion and enhancement transistors have about the same resistance per square of gate area - SPICE shows that this is not a good approximation.

The SPICE simulation gives:

Situation Modelled	SPICE	HAND
sum & carry stable after all cell inputs high	51nS	46nS
x. AND .y stable at mux after y high (x stable)	40nS	34nS
x. AND .y stable at mux after x high (y stable)	30nS	28nS
time to transit mux after inputs inputs high	8nS	12nS
time for X to transit cell (xin to xout)	8nS	12nS
time for Y to transit cell (yin to yout	31nS	20nS
new carry & sum after Y high (x stable)	50nS	46nS
new carry & sum after X high (y stable)	31nS	38nS

Multiplier "Cell"

The multiplier block itself was too large to SPICE-simulate, so we used the data from the SPICE simulation of the leaf adder cell above to calculate that, for our 4x4 implementation, the left carry to output result bit 7 should be available at 3V after 186 nS from the time that all the inputs to the block reach 3V.

This time is very nearly proportional to the number of bits. We note that in a real subsystem, the upper right triangular part of the block (filled

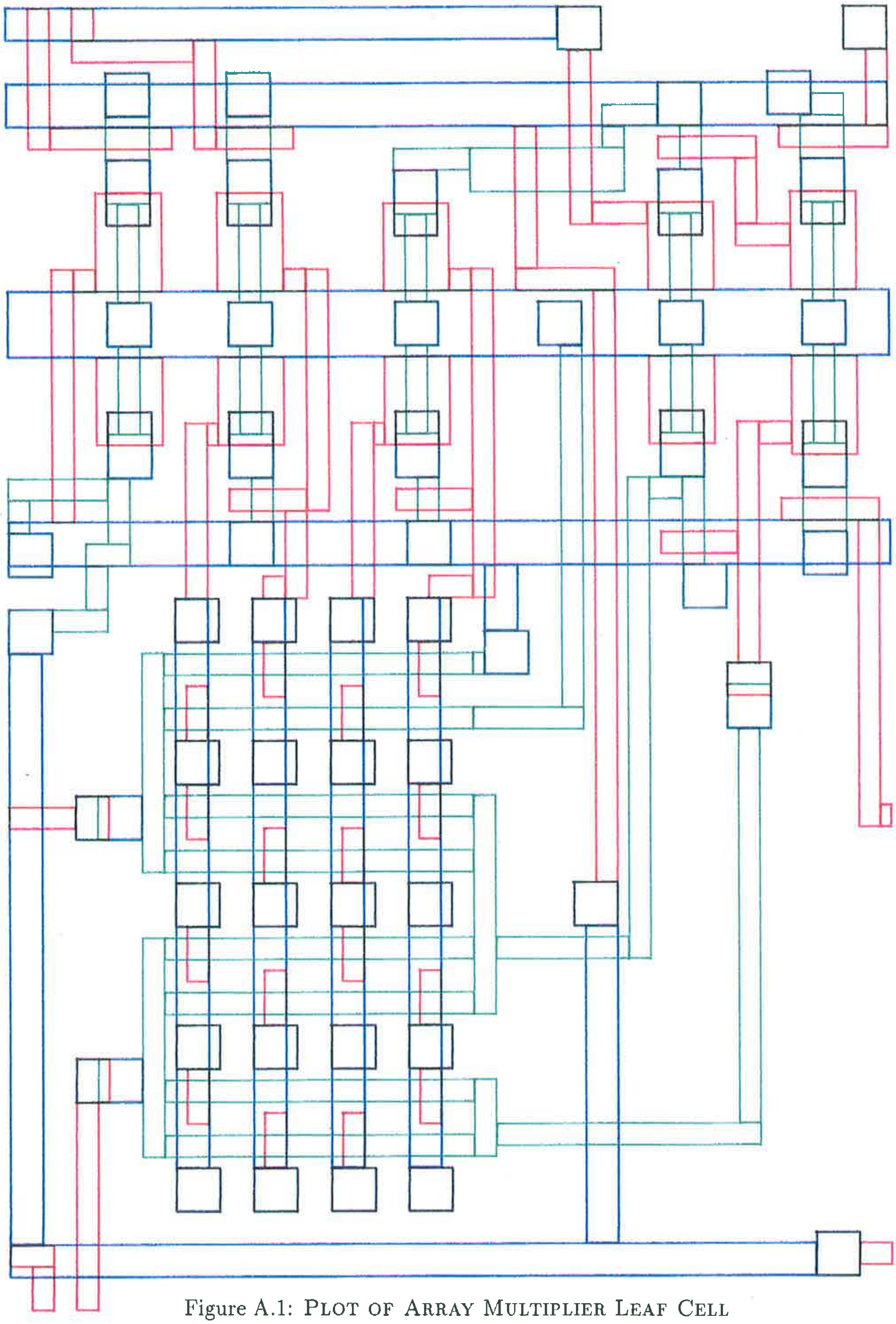


Figure A.1: Plot of Array Multiplier Leaf Cell

with zeros) may be discarded, thus reducing multiply times by 2/3, approximately. Freeney [Fre75] discusses ways of reducing the area by having the same number of output bits as input bits. This involves the bottom right carry-in being tied to +5V. We have left all the cells in this design so that the proportion of time spent multiplying compared to padin/padout time will be large, affording a more accurate measurement of timing for the multiplier block itself.

Adder Leaf Cell

The leaf cell has four inputs: Yin, Xin, Ain and Cin; and four outputs: Yout, Xout, Aout, Cout. It performs the functions:

```
Yout=Yin restored
Xout=Xin restored
C=Cin restored
A=Ain restored
B=Xout.AND.Yout
Aout=low order part of (SUM of A,B,C)
Cout=high order part of (SUM of A,B,C)
```

The multiplexor carries out the full addition of A,B,C. The addition is performed by using the values of A, -A, XY and -XY to turn on pass transistors which allow the signals 0, 1, C or -C to emerge at the ouput to form Aout and Cout. The equations for this are:

$$A_{out} = Z * -C + -Z * C;$$

$$C_{out} = A * XY + Z * C;$$

where

$$Z = A * -XY + -A * XY.$$

The input values are restored by inverters. All power rails are metal. All other connections are polysilicon, except Xin/Xout, which is metal.

Logic Tests

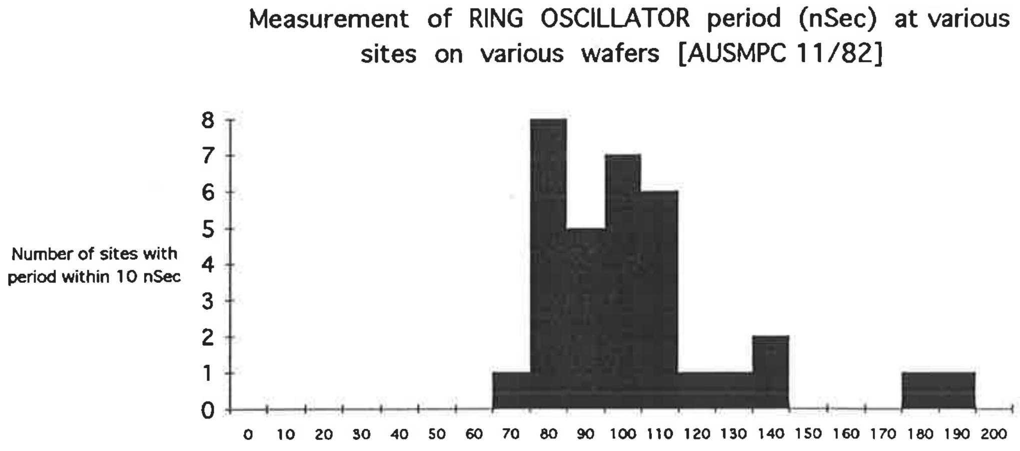
We ran three tests, using MOSSIM [Bry84]:

- around an adder leaf cell in the centre of the multiplication block, where we had four labelled and ten outputs
- for the test adder cell by itself we input test vectors from the pads and monitored the output test pads (ten of them)
- we input a set of X and Y vectors from the pads and monitored the results of the multiplication on the output pads.

  All three tests were completely successful.

A.4 Testing

The bonded chip was submitted to a number of logic and timing tests, under control of a microcomputer, and in conjunction with a CRO. The timing test results[1] made on the ring oscillator, which incidentally first aroused the author's interest in stochastic modelling, are shown below:



[1]effectively the distribution of concatenated finishing times

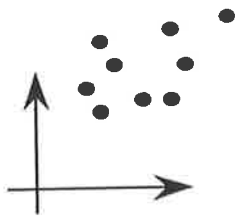
Appendix B

Simulation Technique for Sampling Correlated Variables

This summarizes the usual technique for producing simulations of *correlated* random variables. Since direct generation is not possible, the data are transformed to an axis system where their distributions are uncorrelated (i.e., independent), sampled from these independent distributions, and then transformed back to the original axis system.

Theory

Consider the data points comprising two correlated variables, x and y.



The data are characterised by their *mean values* along each axis

$$< x >= a$$

$$< y >= b$$

and by their *variances*

$$\sigma_{xx}^2 = < (x-a)(x-a) >$$

$$\sigma_{yy}^2 = < (y-b)(y-b) >$$

and by their *covariance*

$$\sigma_{xy} = < (x-a)(y-b) >$$

From these definitions follow the quantities needed later:

$$< xx > = \sigma_{xx}^2 + a^2$$

$$< yy > = \sigma_{yy}^2 + b^2$$

$$< xy > = \sigma_{xy} + ab$$

The degree of *correlation* between the variables affects their covariance; it is measured by the *correlation coefficient, r*, defined as

$$r = \frac{\sigma_{xy}}{\sqrt{\sigma_{xx}\sigma_{yy}}}$$

which varies between -1 (for completely negatively correlated variables) to +1 (for completely positively correlated variables), with zero signifying that the two variables are totally uncorrelated.

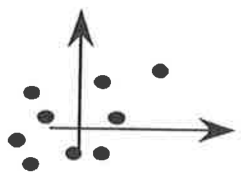
These measures, σ_{xx}, σ_{yy} and σ_{xy}, are properties of the data collection and are unaffected by a change of the origin of the measurement system.

The problem in simulation is to find a set of *uncorrelated* variables that may be used for sampling, i.e., essentially, to make σ_{xy} vanish.

Procedure

This may be done by the following steps:

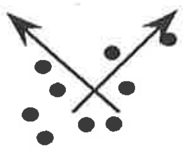
shift the origin so the variables referred to the new axes have zero mean



$$x' = x - a$$

$$y' = y - b$$

rotate the axes so as to make r or σ_{xy}, zero



i.e., rotation by an angle α produces new variables

$$x'' = x' \cos\alpha + y' \sin\alpha$$

$$y'' = -x' \sin\alpha + y' \cos\alpha$$

so that

$$\sigma_{x''y''} = 0$$

i.e.

$$< x''y'' >= 0$$

or

$$< (x' \cos\alpha + y' \sin\alpha)(-x' \sin\alpha + y' \cos\alpha) >= 0$$

or

$$\tan^2\alpha + c\tan\alpha + 1 = 0$$

where

$$c = \frac{< x'x' > - < y'y' >}{< x'y' >}$$

which determines α, the angle of axis rotation.

sample from the distributions, which now have zero means and new independent variances $\sigma^2_{x''x''}$, $\sigma^2_{y''y''}$, calculated from

$$< (x'' - 0)(x'' - 0) >$$

$$=< (x'cos\alpha + y'sin\alpha)(x'cos\alpha + y'sin\alpha) >$$

$$=< x'x' > cos^2\alpha + < y'y' > sin^2\alpha + 2 < x'y' > cos\alpha\, sin\alpha$$

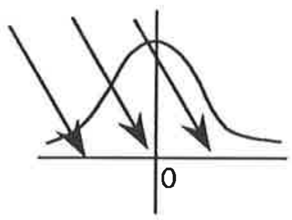
and

$$< (y'' - 0)(y'' - 0) >$$

$$=< x'x' > sin^2\alpha + < y'y' > cos^2\alpha - 2 < x'y' > cos\alpha sin\alpha$$

and of course $< x''y'' >= 0$ since the axis was rotated to ensure this.

Now the data set may be constructed by sampling independently the variables x'' and y'' from distributions with zero means and variances $\sigma^2_{x''x''}$, $\sigma^2_{y''y''}$.



reverse the process to obtain a good representation of the original data, by *unrotating* and *unshifting* the axes, and finally *checking* that the simulated mean and variance closely match the original statistics.

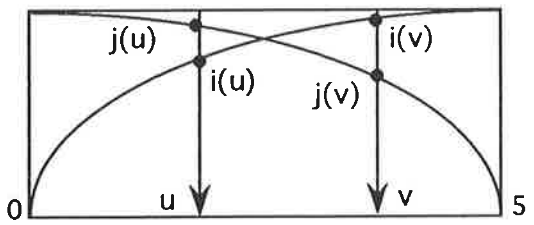
Appendix C

Bargain Equivalence to Newton-Raphson Iteration

Where I prove that my method of using p.e. partner voltages is, in the limit, equivalent to the Newton-Raphson method for finding the root of a function of one variable.

C.1 Proof

Consider the situation using the overlapping current/voltage plots:



We denote the currents calculated in the first π as i and those in the second as j; the voltage assumed at the face in the first is u, and in the second v; the points on the first curve are $i(u)$ and $i(v)$; on the second are $j(u)$ and $j(v)$.

The two straight lines joining the points on each curve are generally:

$$i = A + Bu$$

$$j = C + Dv$$

evaluated at the points shown on the graph we have

$$i(u) = A + Bu$$

$$i(v) = A + Bv$$

$$j(u) = C + Du$$

$$j(v) = C + Dv$$

whence the constants A, B, C, D, provided that u and v differ.

These lines intersect at w, the next guess for the common face voltage, where

$$A + Bw = C + Dw.$$

Inserting the expressions for the constants provides

$$w = \frac{uj(v) - vj(u) + vi(u) - ui(v)}{i(u) - i(v) + j(v) - j(u)}$$

Rearranging this in terms of the current difference

$$k \equiv i - j$$

gives

$$w = \frac{vk(u) - uk(v)}{k(u) - k(v)}$$

and if the voltages differ slightly to the extent that

$$u = v + \delta$$

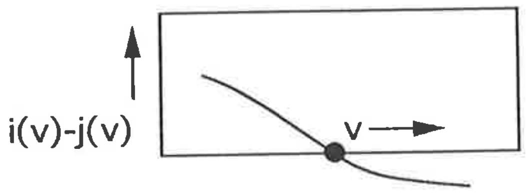
then

$$\begin{aligned} w &= \frac{vk(v+\delta) - (v+\delta)k(v)}{k(v+\delta) - k(v)} \\ &= v - \frac{k(v)}{(k(v+\delta) - k(v))/\delta} \end{aligned}$$

which as $\delta \to 0$ becomes

$$w = v - \frac{k(v)}{k'(v)}$$

This is nothing but the Newton-Raphson algorithm for producing a new guess w from an existing guess v of the root of the function $k(v) \equiv i(v) - j(v)$, i.e., we find v so the face currents are equal.



Note that this works if the function is non-reflexive, which is certainly valid for adjoined transistors, resistors and capacitors.

Appendix D

Details of GAME Program

In this I am concerned with discussing the internal details of the proposed array stochastic modeller.

D.1 Program Description

This appendix is all about the parallel-processing array simulator. It contains all the details needed to understand the program in Appendix E.

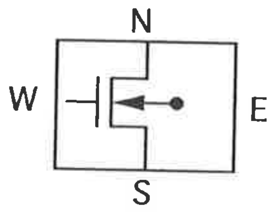
Operating System

The simulator was run on a Sun-4 (SPARC) computer under the BSD 4.2 UNIX operating system which involved command procedures for executing programs written in the 'C' programming language.

Data Structures

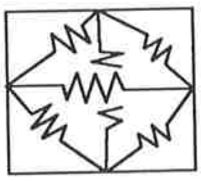
The array to be simulated may contain up to 9 rows and 9 columns of processing elements. Each processing element performs the function of simulating the operation of a single transistor or, where there is no transistor present, a simple lattice of resistors and/or capacitors connected to GND or VDD.

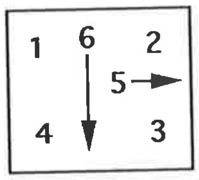
The transistor model is of the form:



with the substrate connected to ground and an option as to which face the gate is connected.

The resistor network is of the form:





which allows current paths to anywhere in the array.

Since I am not concerned with any particular transistor model, but only with whether such an array will work with strictly local rather than global information, then the transistor model is the simplest one that will produce realistic currents for given face voltages. It is

$$I_{DS} = \beta e^{\alpha(V_{GS}+\delta)}(1 - e^{-\alpha V_{DS}})$$

and is adapted from Warner and Grung [WG83].

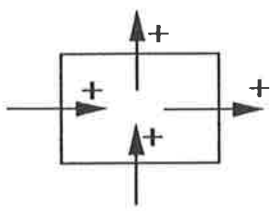
Thus each *p.e.* containing a transistor needs the values of α, β and δ, along with the face to which the gate connects. The parameters in this are α, β and γ, where: α - scales the drain current; β - sets the gain; δ - models depletion mode nMOSFETS.

The resistor network is specified by saying which resistor has which parameters – all resistors have a current of the form

$$i = a + v(b + v(c + v(d)))$$

which is a legacy of early heuristic simulations carried out using non-linear current sources instead of transistors and resistors. The resistor is specified by its number in the *p.e.* (1,2,3,4,5,or 6) and a,b,c,d.

The last thing to notice about the data within a *p.e.* is the sign convention: positive currents are in the direction of the arrows (right and up) on:



the convenience of which will become clear in what follows.

Each *p.e.* is modelled by a C language data type, containing all the information needed by it.

On the outside of the 9 x 9 array there is a ring of dummy processing elements, which are useful both for ensuring that any algorithm works even at the edge of the array, and for entering external driving voltages into the array.

Program Disposition

The simulations I have run are concerned with answering the two questions: Does this model converge to a solution in the quasi- static approximation? How many iterations does it take per timestep, on average, for an entire run?

With these questions in mind, the simulation naturally decomposes into two parts: solving for one timestep, and repeating this over and over while extracting useful results at each timestep.

Thus the core program simply starts with the proposed or actual (or any) set of face voltages, iterates until convergence, and produces the final set of face voltages and currents. From these, plots and summaries may be produced for the whole run.

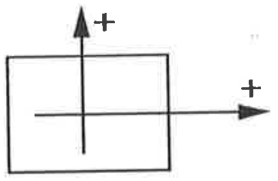
The latter being a relatively trivial operation, only the core operation is dicussed here in some detail.

One Time Step

The purpose of the program listed in the appendix is to take a set of node voltages and produce a new set of node voltages which causes all face currents to balance.

What follows is an implementation of the ideas discussed in the main text of this thesis.

First, we have seen that this is to be a parallel array processor of the SIMD type, and so each processing element executes exactly the same code as all others, at exactly the same instant. There- fore some care has to be taken to ensure that data is exchanged between *p.e.*s in such a way as to be valid - i.e., not to destroy the old data when the new is copied across. This is one reason for the preferred orientation:



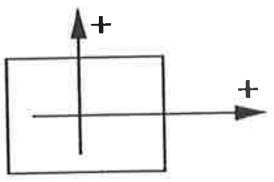
since, if in simulation, *sequentially*, and in reality, *simultaneously*, all data is transferred by all *p.e.*s to the *right* and then *up* into temporary buffers then correct simultaneous operation can be ensured, because these are completely independent operations.

Nomenclature

Now we turn to what is going on inside each *p.e.*, and some definitions are necessary to avoid confusion when referring to the program.

Broadly speaking, given the initial set of all face voltages, each processing element has to do a complete cycle of (normally 4) Newton-Raphson iterations with each neighbour. In practice, the confusion that would cause while trying to keep track of which end-voltages are kept constant would be enough to make debugging very difficult, distracting from the problem itself, which is: does this method work?

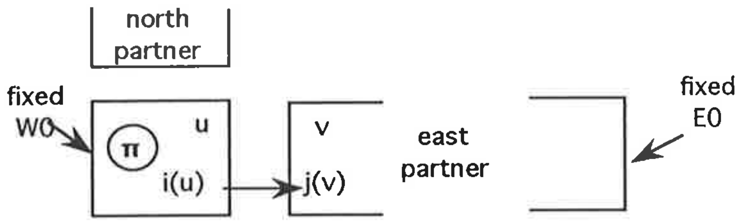
This is where the systematic application of the orientation



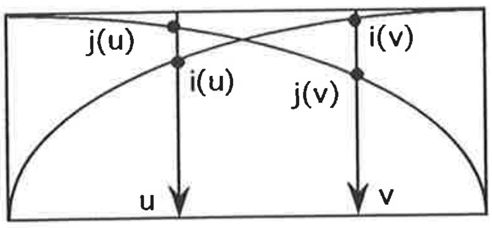
helps, because each *p.e.* really only needs to deal completely with its North and East neighbours - its West and South iterations are then taken care of by the West and South neighbours - this is particularly vital at the edge of the array (left, bottom).

Right and North Iteration

Concentrating, then, as we may, only on these two neighbours which we now call "the north partner" and "the east partner" of the *p.e.*, and recalling the current calculations necessary for producing local convergence at one face, the situation is:



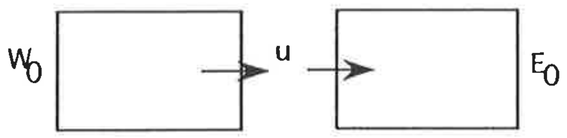
Now recall that to complete one Newton-Raphson iteration the *p.e.* only needs the 4 numbers representing the points on the graph:



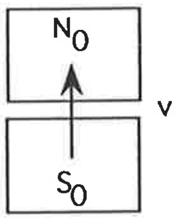
and it obtains 2 of them from within itself – $i(u)$ and $i(v)$ – and the other 2 – $j(u)$ and $j(v)$ – from its relevant partner (N or E). This means that it has to pass these values to the left and down, in its turn.

Also remember from the study of the Newton-Raphson convergence, that while the 4 iterations proceed, the voltages at the outermost ends of the relevant *p.e.* 'pair' (the *p.e.* and its $N-$ or $E-$ partner) must remain constant.

These are the voltages W_0 and E_0 below, whilst u is being determined,

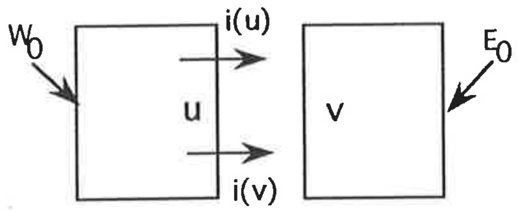


or the voltages N_0 and S_0 whilst v is determined, below



Thus at the start of the 4 Newton-Raphson iterations, the face voltages need to be preserved, only being updated at the end of the 4 iterations, the 'preserved' values being used where appropriate in the calculations.

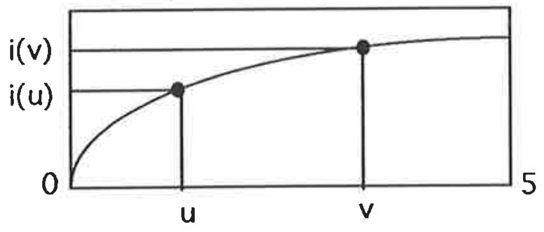
The notation for the four values required by each *p.e.* is now to be settled.



Consider the *p.e.* pair above.

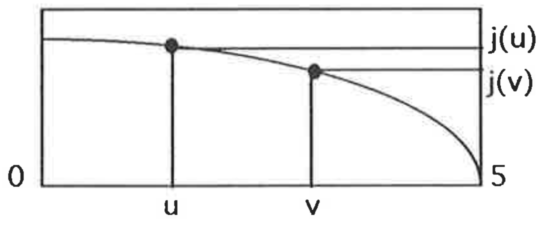
The currents within the *p.e.* calculated using u and v are: $i(u)$ and $i(v)$.

These correspond to the points on the superimposed plot for the pair:

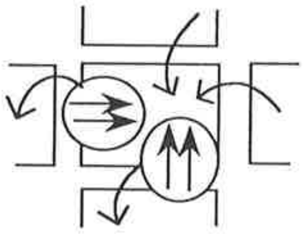


The currents calculated <u>in</u> its partner <u>by</u> its partner are denoted $j(u)$ and

$j(v)$ corresponding to the points on the conjugate plot



Thus in each set of processing elements,



the West and South values are transferred in one go.

The Iteration Algorithm

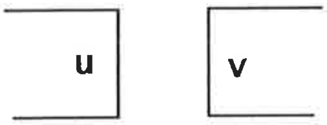
The complete Newton-Raphson sequence is thus:

for each timestep

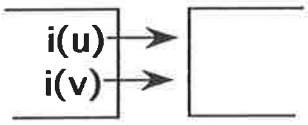
for each face-pair (North, East)

- establish the voltage common to the *p.e.*s at the common face - call it u;

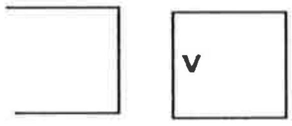
- force them to hold slight different values $u \pm \delta = u, v$



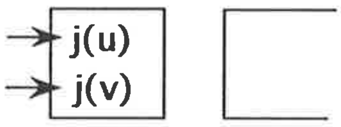
- compute internal currents $i(u)$ and $i(v)$, keeping all other face voltages constant (i.e at their original values);



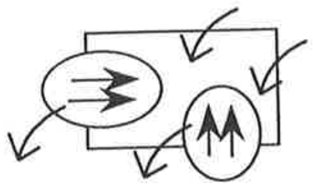
- establish the voltage at the partner's face - call it $v = u + \delta$;



- compute the internal currents to be passed left and down - $j(u)$ and $j(v)$;



- put *(to left and down)*, or get *(from right and up)*, the currents from the partner element via the relevant face.



- compute the new estimate of the voltage at the common face and share it with partner (right and up), being careful of the edge driving voltages.

This takes us towards an economical and systematic notation, which is the same as that employed in the program listing.

The exact correspondence between this scheme and the 'C' program is shown in these scheme:

$$\pi(p).i_N(\alpha) \leftrightarrow i.n$$

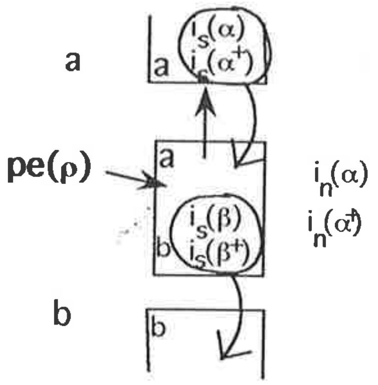
$$\pi(p).i_N(\alpha^+) \leftrightarrow ip.n$$

$$\pi(p+1).i_S(\alpha) \leftrightarrow j.n$$

$$\pi(p+1).i_S(\alpha^+) \leftrightarrow jp.n$$

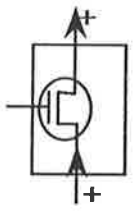
Since each *p.e.* only calculates for the East and North faces, then this minimum notation is sufficient.

This notation relates to the diagram, with inner and outer sets of parameters:

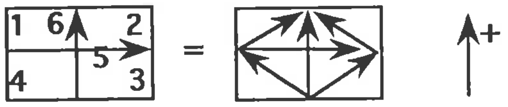


One further note on the sign of currents:

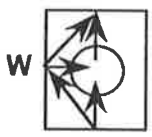
For transistors:



For resistors:



or both can be present:

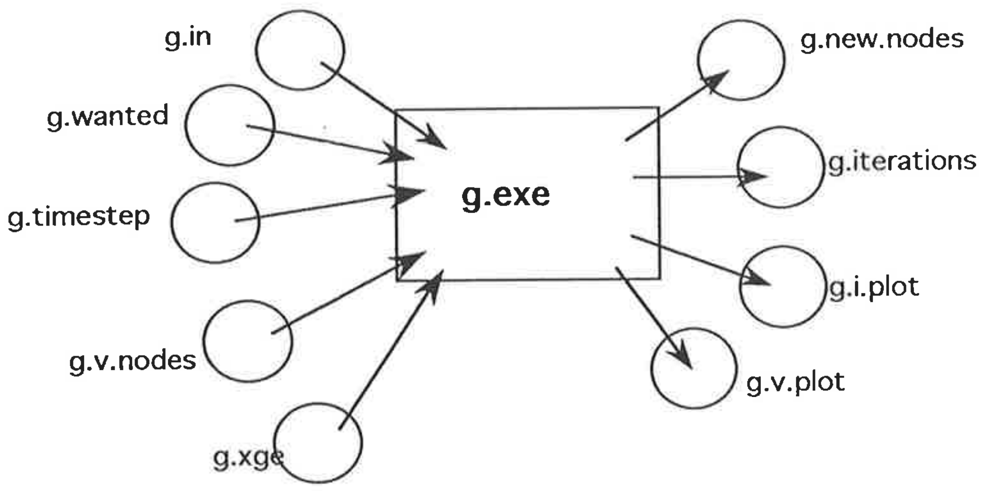


and this is reflected in the program – for example, the west current:

$$iw = +i1 + i5 - i4.$$

Running the Simulator One Time Step

The operating system is:



where the files used are:

g.in = general inputs, such as the contents of each *p.e.*, how many iterations to try and the criteria for convergence;

g.timestep = this timestep;

g.wanted = list of voltage nodes for plotting;

g.v.modes = set of face voltages at start of timestep.

g.new.nodes = set of face voltages at end of timestep, with voltages and currents within specified tolerance;

g.iterations = number of iterations to converge;

g.i.plot = face currents at each iteration for plot;

g.v.plot = face voltages at each iteration for plot.

g.xge = exchange rates for each array element.

Input/Output Files

These are examples of the main files used for the run involving the flip-flop.

g.in

```
53      number of bits
100     reporting interval
1 5 1 5 1 5 1 5  s1xf1 s2xf2 s3xf3 s4xf4
5000     grand iterations
1000.0  current limit (uA)
0.1      dt (uSec)
0.05     delta voltage (V)
5.0      vss (V)
```

g.circuit

```
1 2 7 1 8 1.5 s
1 7 7 1 8 1.5 s
2 2 6 100
2 7 6 100
3 2 6 100
3 7 6 100
4 2 6 100
4 7 6 100
5 2 6 100
5 7 6 100
6 2 3 100
6 2 6 100
6 3 4 100
6 4 3 100
6 5 5 100
6 6 5 100
6 7 4 100
6 7 6 100
7 1 3 100
7 2 1 100
7 2 2 100
7 3 4 100
7 3 2 100
7 4 5 100
7 4 6 100
7 5 4 100
7 6 3 100
7 6 8 10 10 se00
7 7 1 100
7 7 2 100
7 8 4 100
8 4 1 100
8 5 2 100
8 1 7 1 2 0 w
8 3 7 1 2 0 e
8 6 7 1 2 0 w
8 8 7 1 2 0 e
0 0 0
```

g.wanted

```
w 8 1
e 8 8
n 7 2
n 7 7
0
```

g.xge

```
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
 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
 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
 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
```

g.nodes

```
0 5 0 0 0 0 5 0
0 0 0 0 0 0 0 0 0
0 3 0 0 0 0 4 0
0 0 0 0 0 0 0 0 0
0 2 0 0 0 0 4 0
0 0 0 0 0 0 0 0 0
0 2 0 0 0 0 4 0
0 0 0 0 0 0 0 0 0
0 2 0 0 0 0 3 0
0 0 0 0 0 0 0 0 0
0 1 0 0 0 0 3 0
0 0 1 0 3 3 3 0 0
0 1 1 3 0 0 3 0
0 1 2 1 1 0 3 3 0
1 0 0 3 1 3 0 3
0 0 0 3 0 1 0 0 0
0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0
 0 0 0 0 0 3 0 0
 0 0 0 0 0 0 0 0
```

Appendix E

GAME Program Listing

```
 /*  program  GAME.C  */

#include <stdio.h>
#include <math.h>
                              /* masks for double floating point numbers */
unsigned char maskset1[8]={255,240,000,000,000,000,000,000};
unsigned char maskset2[8]={255,248,000,000,000,000,000,000};
unsigned char maskset3[8]={255,252,000,000,000,000,000,000};
unsigned char maskset4[8]={255,254,000,000,000,000,000,000};
unsigned char maskset5[8]={255,255,000,000,000,000,000,000};
unsigned char maskset6[8]={255,255,128,000,000,000,000,000};
unsigned char maskset7[8]={255,255,192,000,000,000,000,000};
unsigned char maskset8[8]={255,255,224,000,000,000,000,000};
unsigned char maskset9[8]={255,255,240,000,000,000,000,000};
unsigned char maskset10[8]={255,255,248,000,000,000,000,000};

unsigned char maskset11[8]={255,255,252,000,000,000,000,000};
unsigned char maskset12[8]={255,255,254,000,000,000,000,000};
unsigned char maskset13[8]={255,255,255,000,000,000,000,000};
unsigned char maskset14[8]={255,255,255,128,000,000,000,000};
unsigned char maskset15[8]={255,255,255,192,000,000,000,000};
unsigned char maskset16[8]={255,255,255,224,000,000,000,000};
unsigned char maskset17[8]={255,255,255,240,000,000,000,000};
unsigned char maskset18[8]={255,255,255,248,000,000,000,000};
unsigned char maskset19[8]={255,255,255,252,000,000,000,000};
unsigned char maskset20[8]={255,255,255,254,000,000,000,000};

unsigned char maskset21[8]={255,255,255,255,000,000,000,000};
unsigned char maskset22[8]={255,255,255,255,128,000,000,000};
unsigned char maskset23[8]={255,255,255,255,192,000,000,000};
unsigned char maskset24[8]={255,255,255,255,224,000,000,000};
unsigned char maskset25[8]={255,255,255,255,240,000,000,000};
unsigned char maskset26[8]={255,255,255,255,248,000,000,000};
unsigned char maskset27[8]={255,255,255,255,252,000,000,000};
unsigned char maskset28[8]={255,255,255,255,254,000,000,000};
unsigned char maskset29[8]={255,255,255,255,255,000,000,000};
unsigned char maskset30[8]={255,255,255,255,255,128,000,000};

unsigned char maskset31[8]={255,255,255,255,255,192,000,000};
unsigned char maskset32[8]={255,255,255,255,255,224,000,000};
unsigned char maskset33[8]={255,255,255,255,255,240,000,000};
unsigned char maskset34[8]={255,255,255,255,255,248,000,000};
unsigned char maskset35[8]={255,255,255,255,255,252,000,000};
unsigned char maskset36[8]={255,255,255,255,255,254,000,000};
unsigned char maskset37[8]={255,255,255,255,255,255,000,000};
unsigned char maskset38[8]={255,255,255,255,255,255,128,000};
unsigned char maskset39[8]={255,255,255,255,255,255,192,000};
unsigned char maskset40[8]={255,255,255,255,255,255,224,000};

unsigned char maskset41[8]={255,255,255,255,255,255,240,000};
unsigned char maskset42[8]={255,255,255,255,255,255,248,000};
unsigned char maskset43[8]={255,255,255,255,255,255,252,000};
unsigned char maskset44[8]={255,255,255,255,255,255,254,000};
unsigned char maskset45[8]={255,255,255,255,255,255,255,000};
unsigned char maskset46[8]={255,255,255,255,255,255,255,128};
unsigned char maskset47[8]={255,255,255,255,255,255,255,192};
unsigned char maskset48[8]={255,255,255,255,255,255,255,224};
unsigned char maskset49[8]={255,255,255,255,255,255,255,240};
unsigned char maskset50[8]={255,255,255,255,255,255,255,248};

unsigned char maskset51[8]={255,255,255,255,255,255,255,252};
unsigned char maskset52[8]={255,255,255,255,255,255,255,254};
unsigned char maskset53[8]={255,255,255,255,255,255,255,255};
```

```
#define ROWS 8
#define COLS 8
#define TRUE 1
#define FALSE 0

struct currentstruct
            {
            double n;
            double e;
            double w;
            double s;
            };

struct voltagestruct
            {
            double n;
            double e;
            double w;
            double s;
            };

struct capacitorstruct
            {
            double cn,ce,cw,cs;
        double c;
        double r;
        double v;
        double jn,je,jw,js;
        double dv;
            };

struct wantedstruct
{
            int n,e,w,s;
};

struct pricestruct
{
double n,e,w,s;
};

struct quantitystruct
{
double n,e,w,s;
};

struct outlaystruct
{
double n,e,w,s;
};

struct pestruct
{
struct voltagestruct v;
double conductance[7];
double alpha;
double beta;
double delta;
double volts[7];
double iamps[7];
double sign[7];
double igate;
double idrain;
char gateconnection;
struct capacitorstruct cap;
struct currentstruct i;
struct currentstruct j;
struct currentstruct ip;
struct currentstruct jp;
struct voltagestruct v0;
struct currentstruct i0;
int done;
struct voltagestruct new;
struct currentstruct residual;
struct pricestruct price[3];/* use only 1 and 2*/
double exchangerate;
double profit[3][3][3][3];/*use only 1 and 2*/
};

unsigned char bitmask[8];
int masking;     /*TRUE if bits valid (between 1 and 53)*/
double multiply();
double divide();
double add();
double subtract();
double expf();
double fetch();
double store();
double absolute();
int bits;
int numberadd;
int numbersubtract;
int numbermultiply;
int numberdivide;
int numberfetch;
int numberstore;
int numberabsolute;

crlf();
int alldone();
double modeln();
double modele();
double modelw();
double models();
double largestresidual();

FILE *arrayfile;
FILE *inputfile;
FILE *outputfile;
FILE *nodefile;
FILE *exchangefile;
FILE *wantedfile;
FILE *circuitfile;
FILE *plotfile;
FILE *operationsfile;

int iterations;
int granditerations;
struct pestruct pe[ROWS+2][COLS+2];
struct wantedstruct wanted[ROWS+2][COLS+2];
double currentlimit;
double deltavoltage;
double dt;
double vss;
int plotting;
int progress;
int reportinterval;
int applygamecriterion[1000];/* boolean at which iteration to apply game
theory*/
int gameintervalstart1;
int gameintervalstart2;
int gameintervalstart3;
int gameintervalstart4;
int gamefrequency1;
int gamefrequency2;
int gamefrequency3;
int gamefrequency4;
double capjn,capje,capjw,capjs;
char string[80];


main()
{
printf("\nprogram GAME started\n");

inputfile=fopen("g.in","r");
if (inputfile==NULL)  {printf("WHERE IS G.IN???\n"); exit(8);}
printf("input data read from G.IN\n");
fscanf(inputfile,"%d",&bits); fgets(string,80,inputfile);
masking=TRUE;            /*default*/
 if ( (bits<1) || (bits>53) )
          {bits=53;
          masking=FALSE;}
if (masking==TRUE)
{
setupbitmask(bits);
printf("using bit mask for %d bits: %d %d %d %d %d %d %d
%d\n",bits,bitmask[0],bitmask[1],bitmask[2],bitmask[3],bitmask[4],bitmask[5],
bitmask[6],bitmask[7]);
}
else printf("53 bits, no masking\n");

initialise();
gameheading();
iterations=0;
progress=0;
plotdata();
do
{
iterations=iterations+1;
progress=progress+1;
if (progress>=reportinterval) reportprogress();
iterate();
plotdata();
}
while ( (iterations<granditerations) && !(alldone()) );
reportprogress();
computecapacitorincrement();
savenodes();
savenodesininputformat();      /*for passing to bargain*/
if (masking==TRUE)
{
saveoperations();
printf("processor operations per iteration:\n");
printops(numbermultiply); printf(" multiplications\n");
printops(numberdivide); printf(" divisions\n");
printops(numberadd); printf(" additions\n");
printops(numbersubtract); printf(" subtractions\n");
}
```

```
printf("total of %4d iterations taken using %2d bits\n",iterations,bits);
if (masking==TRUE)
{
        printf("total +-x/ operations (index of total time) =");
        index(numberadd+numbersubtract+numbermultiply+numberdivide);
}
printf("program GAME finished\n");

}


index(total)
int total;
{
if ( (total<1000) )
        printf("%4.1f\n",(float)total);
if ( (total>=1000) && (total<1000000) )
        printf("%4.1f thousand\n",(float)total/1000.0);
if ((total>=1000000) && (total<1000000000) )
        printf("%6.1f million\n",(float)total/1000000.0);
}


printops(operations)
int operations;
{
extern int iterations;
printf("%6.0f thousand",(float)operations/(float)iterations/1000.0);
}


getwanted()
{
extern int plotting;
int r,c;
char face;
char carriagereturn;

/* set all default wanted to false*/
  plotting=FALSE;
  for (r=1; r<=ROWS; r=r++)
  {
    for (c=1; c<=COLS; c=c++)
    {
      wanted[r][c].n=FALSE;
      wanted[r][c].e=FALSE;
      wanted[r][c].w=FALSE;
      wanted[r][c].s=FALSE;
    }
  }

wantedfile=fopen("g.wanted","r");
if (wantedfile==NULL) {printf("WHERE IS G.WANTED???\n"); exit(7);}
printf("plots from G.WANTED: ");
fscanf(wantedfile,"%c",&face);
while ( (face=='n') || (face=='e') || (face=='w') || (face=='s') )
{
plotting=TRUE;
printf("%c",face);
fscanf(wantedfile,"%d %d",&r,&c);
printf("%1d%1d ",r,c);
switch(face)
{
```

```
case 'n': wanted[r][c].n=TRUE; break;
case 'e': wanted[r][c].e=TRUE; break;
case 'w': wanted[r][c].w=TRUE; break;
case 's': wanted[r][c].s=TRUE; break;
default: printf("ERROR IN G.WANTED\n"); exit(6);
}
fscanf(wantedfile,"%c %c",&carriagereturn,&face);
}

if (plotting==FALSE) printf("(none)\n");
else
{
plotfile=fopen("g.plot","w");
printf("\nplotting to G.PLOT in excel format\n");
fprintf(plotfile,"iter.\t maxres\t ");

for (r=1;r<=ROWS;r++)
{
for (c=1;c<=8; c++)
{
if (wanted[r][c].n==TRUE) fprintf(plotfile,"    n%1d%1d\t ",r,c);
if (wanted[r][c].e==TRUE) fprintf(plotfile,"    e%1d%1d\t ",r,c);
if (wanted[r][c].w==TRUE) fprintf(plotfile,"    w%1d%1d\t ",r,c);
if (wanted[r][c].s==TRUE) fprintf(plotfile,"    s%1d%1d\t ",r,c);
}
}
fprintf(plotfile,"\n");
}
}


initialise()
{
extern int
granditerations,
reportinterval,
applygamecriterion[1000],
gameintervalstart1,
gameintervalstart2,
gameintervalstart3,
gameintervalstart4,
gamefrequency1,
gamefrequency2,
gamefrequency3,
gamefrequency4;
extern double dt,
currentlimit,
delatvolt,
vss;

FILE *appliedfile;
int r,c,i;
int dummy;
int p;
char f1,f2,f3,f4;

        fscanf(inputfile,"%d",&reportinterval);fgets(string,80,inputfile);
        fscanf(inputfile,"%d %d  %d %d  %d %d  %d %d",
                &gameintervalstart1,
                &gamefrequency1,
                &gameintervalstart2,
                &gamefrequency2,
                &gameintervalstart3,
                &gamefrequency3,
                &gameintervalstart4,
                &gamefrequency4);fgets(string,80,inputfile);
        fscanf(inputfile,"%d",
                &granditerations);fgets(string,80,inputfile);
        fscanf(inputfile,"%lf",
                &currentlimit); fgets(string,80,inputfile);
        fscanf(inputfile,"%lf",
                &dt); fgets(string,80,inputfile);
        fscanf(inputfile,"%lf",
                &deltavoltage); fgets(string,80,inputfile);
        fscanf(inputfile,"%lf",
                &vss); fgets(string,80,inputfile);
        currentlimit=currentlimit/1000000.0;
        dt=dt/1000000000.0;
        printf("%d report interval\ngame from %d x %d; from %d x %d; from
%d x %d; from %d x %d\n%d max. iterations\n",
                reportinterval,
                gameintervalstart1,
                gamefrequency1,
                gameintervalstart2,
                gamefrequency2,
                gameintervalstart3,
                gamefrequency3,
                gameintervalstart4,
                gamefrequency4,
                granditerations);
printf("%6.1f uA current convergence criterion\n",(1000000.0*currentlimit));
printf("%6.1f nSec time step\n",(1000000000.0*dt));
printf("%4.2f deltavoltage\n %3.1f vss\n",deltavoltage,vss);

/* at each frequency wanted to apply game theory, put a TRUE marker*/
for (p=1;p<1000;p++) applygamecriterion[p]=FALSE;
for (p=gameintervalstart1;(p>0 && p<1000);p=p+gamefrequency1)
applygamecriterion[p]=TRUE;
for (p=gameintervalstart2;(p>0 && p<1000);p=p+gamefrequency2)
applygamecriterion[p]=TRUE;
for (p=gameintervalstart3;(p>0 && p<1000);p=p+gamefrequency3)
applygamecriterion[p]=TRUE;
for (p=gameintervalstart4;(p>0 && p<1000);p=p+gamefrequency4)
applygamecriterion[p]=TRUE;
/* write out for referece*/
appliedfile=fopen("applied.dat","w");
fprintf(appliedfile,"game criterion applied at iterations:\n");
for (p=1;p<1000;p++)
    if (applygamecriterion[p]==TRUE)
        fprintf(appliedfile,"%4d\n",p);
fclose(appliedfile);
printf("game criterion applications to APPLIED.DAT\n");

/* read array data for whole pe */

arrayfile=fopen("array.dat","r");
if (arrayfile==NULL)
      {printf("ERROR GETTING ARRAYFILE ARRAY.DAT\n"); exit(1);}

fread(pe,sizeof(struct pestruct),(ROWS+2)*(COLS+2),arrayfile);
fclose(arrayfile);
if (ferror(arrayfile)!=0) {printf("pe array not read in\n"); exit(13);}
getwanted();
clearnumbers();
}
```

```
plotdata()
{
extern int iterations;
extern int numberadd;
extern int numbersubtract;
extern int numbermultiply;
extern int numberdivide;
int r,c;

if (plotting==TRUE)
{
  fprintf(plotfile,
            "%4d \t %10.0f \t "     ,
            iterations,(1000000.0*largestresidual()));
      fprintf(plotfile,
            "%12d \t",

      (numberadd+numbersubtract+numbermultiply+numberdivide));
  for (r=1;r<=ROWS;r++)
  {
   for (c=1;c<=COLS; c++)
   {
    if (wanted[r][c].n==TRUE) fprintf(plotfile,"%12.9f \t",pe[r][c].v.n);
    if (wanted[r][c].e==TRUE) fprintf(plotfile,"%12.9f \t",pe[r][c].v.e);
    if (wanted[r][c].w==TRUE) fprintf(plotfile,"%12.9f \t",pe[r][c].v.w);
    if (wanted[r][c].s==TRUE) fprintf(plotfile,"%12.9f \t",pe[r][c].v.s);
    if (wanted[r][c].n==TRUE) fprintf(plotfile,"%15.12lf \t",pe[r][c].i.n);
    if (wanted[r][c].e==TRUE) fprintf(plotfile,"%15.12lf \t",pe[r][c].i.e);
    if (wanted[r][c].w==TRUE) fprintf(plotfile,"%15.12lf \t",pe[r][c].i.w);
    if (wanted[r][c].s==TRUE) fprintf(plotfile,"%15.12lf \t",pe[r][c].i.s);
   }
  }
 fprintf(plotfile,"\n");
 }
}


saveoperations()
{
extern int numberadd;
extern int numbersubtract;
extern int numbermultiply;
extern int numberdivide;
extern int numberabsolute;
extern int numberfetch;
extern int numberstore;
int r,c;

operationsfile=fopen("g.operations","w");
printf("operation count to G.OPERATIONS\n");
fprintf(operationsfile,"absolute\t%12d\n ",numberabsolute);
fprintf(operationsfile,"add     \t%12d\n ",numberadd);
fprintf(operationsfile,"divide  \t%12d\n ",numberdivide);
fprintf(operationsfile,"fetch   \t%12d\n ",numberfetch);
fprintf(operationsfile,"multiply\t%12d\n ",numbermultiply);
fprintf(operationsfile,"store   \t%12d\n ",numberstore);
fprintf(operationsfile,"subtract\t%12d\n ",numbersubtract);
fclose(operationsfile);
}
```

```
clearnumbers()
{
numberadd=0;
numbersubtract=0;
numbermultiply=0;
numberdivide=0;
numberfetch=0;
numberstore=0;
numberabsolute=0;
}


allcurrents(r,c,vn,ve,vw,vs)
int r,c;
double vn,ve,vw,vs;
{
extern double capjn,capje,capjw,capjs;
int i;
double vgs;
double vds;
struct pestruct *p;
p=&pe[r][c];
p->volts[1]=store(subtract(fetch(vn),fetch(vw)));
                    if (fetch(p->volts[1])>0) p->sign[1]=store(1.0); else p-
>sign[1]=store(-1.0);
p->volts[2]=store(subtract(fetch(vn),fetch(ve)));
                    if (fetch(p->volts[2])>0) p->sign[2]=store(1.0); else p-
>sign[2]=store(-1.0);
p->volts[3]=store(subtract(fetch(ve),fetch(vs)));
                    if (fetch(p->volts[3])>0) p->sign[3]=store(1.0); else p-
>sign[3]=store(-1.0);
p->volts[4]=store(subtract(fetch(vw),fetch(vs)));
                    if (fetch(p->volts[4])>0) p->sign[4]=store(1.0); else p-
>sign[4]=store(-1.0);
p->volts[5]=store(subtract(fetch(ve),fetch(vw)));
                    if (fetch(p->volts[5])>0) p->sign[5]=store(1.0); else p-
>sign[5]=store(-1.0);
p->volts[6]=store(subtract(fetch(vn),fetch(vs)));
                    if (fetch(p->volts[6])>0) p->sign[6]=store(1.0); else p-
>sign[6]=store(-1.0);

if (p->gateconnection=='n') vgs=store(subtract(fetch(vn),fetch(vs)));
else
if (p->gateconnection=='e') vgs=store(subtract(fetch(ve),fetch(vs)));
else
if (p->gateconnection=='w') vgs=store(subtract(fetch(vw),fetch(vs)));
else
if (p->gateconnection=='s') vgs=store(subtract(fetch(vs),fetch(vs)));
else vgs=store(subtract(fetch(vw),fetch(vs)));
vds=store(subtract(fetch(vn),fetch(vs)));

for (i=1; i<=6; i=i++) p->iamps[i]=
          store(   divide(
                              multiply(
                                        fetch(p->sign[i]),
                                        multiply(
                                                  absolute(fetch(p->volts[i])),
                                                  fetch(p->conductance[i])
                                                  )
                                        ),
                              fetch(1000.0)
                              )
                    );
```

```
p->igate=store(0.0);
if (vds>0) p->idrain=
        store(
        (
        (fetch(p->beta)*
        (expf(
                multiply(
                        fetch(p->alpha),
                        add(    fetch(vgs),
                                fetch(p->delta)
                                )
                        )

        )*
        (fetch(1.0)-
                expf(
                        multiply(-fetch(p->alpha),
                                fetch(vds)
                                )
                )
        )
        )
        )
        /fetch(1000.0))
        );
else p->idrain=store(0.0);
capjn=store(multiply(  (fetch(vn)-fetch(p->cap.v)),
                fetch(p->cap.cn)
                )
        );
capje=store(multiply(  (fetch(ve)-fetch(p->cap.v)),
                fetch(p->cap.ce)
                )
        );
capjw=store(multiply(           (fetch(p->cap.v)-fetch(vw)),
                fetch(p->cap.cw)
                )
        );
capjs=store(multiply(  (fetch(p->cap.v)-fetch(vs)),
                fetch(p->cap.cs)
                )
        );

p->cap.jn=store(capjn);
p->cap.je=store(capje);
p->cap.jw=store(capjw);
p->cap.js=store(capjs);
}


double   modeln(r,c,vn,ve,vw,vs)
int r,c;
double vn,ve,vw,vs;
{
extern double capjn;
struct pestruct *p;
p=&pe[r][c];
        allcurrents(r,c,vn,ve,vw,vs);

        return
        add(
                fetch(p->iamps[1]),
                add(
                        fetch(p->iamps[6]),
                        add(
                                fetch(p->iamps[2]),
                                add(
                                        fetch(p->idrain),
                                        fetch(capjn)
                                )
                        )
                )
        );
}


double  modele(r,c,vn,ve,vw,vs)
int r,c;
double vn,ve,vw,vs;
{
extern double capje;
struct pestruct *p;
p=&pe[r][c];

        allcurrents(r,c,vn,ve,vw,vs);

        return add(
                        -fetch(p->iamps[2]),
                        add(
                                fetch(p->iamps[5]),
                                add(
                                        fetch(p->iamps[3]),
                                        fetch(capje)
                                )
                        )
                );
}


double  modelw(r,c,vn,ve,vw,vs)
int r,c;
double vn,ve,vw,vs;
{
extern double capjw;
struct pestruct *p;
p=&pe[r][c];
        allcurrents(r,c,vn,ve,vw,vs);
        return add(
                        fetch(p->iamps[1]),
                        add(
                                fetch(p->iamps[5]),
                                add(
                                        -fetch(p->iamps[4]),
                                        add(
                                                fetch(p->igate),
                                                fetch(capjw)
                                        )
                                )
                        )
                );
}
```

```
double   models(r,c,vn,ve,vw,vs)
int r,c;
double vn,ve,vw,vs;
{
extern double capjs;
struct pestruct *p;
p=&pe[r][c];
          allcurrents(r,c,vn,ve,vw,vs);
          return add(
                              fetch(p->iamps[4]),
                              add(
                                        fetch(p->iamps[6]),
                                        add(
                                                  fetch(p->iamps[3]),
                                                  add(
                                                            fetch(p->idrain),
                                                            fetch(capjs)
                                                  )
                                        )
                              )
                    );
}


computecurrents(r,c)
int r,c;
{
extern double deltavoltage;
struct pestruct *p;
p=&pe[r][c];
p->i.n=store(modeln(r,c,fetch(p->v.n),
                              fetch(p->v0.e),
                              fetch(p->v0.w),
                              fetch(p->v0.s))
                    );
p->ip.n=store(modeln(r,c,add(fetch(p->v.n),fetch(deltavoltage)),
                              fetch(p->v0.e),
                              fetch(p->v0.w),
                              fetch(p->v0.s))
                    );
p->i.s=store(models(r,c,fetch(p->v0.n),
                              fetch(p->v0.e),
                              fetch(p->v0.w),
                              fetch(p->v.s))
                    );
p->ip.s=store(models(r,c,fetch(p->v0.n),
                              fetch(p->v0.e),
                              fetch(p->v0.w),
                              add(fetch(p->v.s),fetch(deltavoltage)))
                    );

p->i.e=store(modele(r,c,fetch(p->v0.n),
                              fetch(p->v.e),
                              fetch(p->v0.w),
                              fetch(p->v0.s)));
p->ip.e=store(modele(r,c,fetch(p->v0.n),
                              add(fetch(p->v.e),fetch(deltavoltage)),
                              fetch(p->v0.w),
                              fetch(p->v0.s)));
p->i.w=store(modelw(r,c,fetch(p->v0.n),
                              fetch(p->v0.e),
                              fetch(p->v.w),
                              fetch(p->v0.s)));
p->ip.w=store(modelw(r,c,fetch(p->v0.n),
                          fetch(p->v0.e),
                          add(fetch(p->v.w),fetch(deltavoltage)),
                          fetch(p->v0.s)));
}
```

maxprofitcriterion(r,c)

```
int r,c;
{
double maxprofit;
int nmax,emax,wmax,smax;
int n,e,w,s;
  maxprofit=store(-10000000.0);
  nmax=2;
  emax=2;
  wmax=2;
  smax=2;
  for (n=1; n<=2; n++)
    {for (e=1; e<=2; e++)
      {for (w=1; w<=2; w++)
        {for (s=1; s<=2; s++)
          {
            if (fetch(pe[r][c].profit[n][e][w][s])>=fetch(maxprofit))
             {
              maxprofit=store(pe[r][c].profit[n][e][w][s]);
              nmax=n;
              emax=e;
              wmax=w;
              smax=s;
             }
          }
        }
      }
    }
               pe[r][c].v.n=store(pe[r][c].price[nmax].n);
               pe[r][c].v.e=store(pe[r][c].price[emax].e);
               pe[r][c].v.w=store(pe[r][c].price[wmax].w);
               pe[r][c].v.s=store(pe[r][c].price[smax].s);

}
```

computepossibles(r,c)

```
int r,c;
{
int n,e,w,s;
struct quantitystruct quantity;
struct outlaystruct outlay;
        for (n=1; n<=2; n++)
        {for (e=1; e<=2; e++)
        {for (w=1;w<=2;w++)
        {for (s=1;s<=2;s++)
                {quantity.n=store(modeln(r,c,fetch(pe[r][c].price[n].n),
                                        fetch(pe[r][c].price[e].e),
                                        fetch(pe[r][c].price[w].w),
                                        fetch(pe[r][c].price[s].s)));
                quantity.e=store(modele(r,c,fetch(pe[r][c].price[n].n),
                                        fetch(pe[r][c].price[e].e),
                                        fetch(pe[r][c].price[w].w),
                                        fetch(pe[r][c].price[s].s)));
                quantity.w=store(modelw(r,c,fetch(pe[r][c].price[n].n),
                                        fetch(pe[r][c].price[e].e),
                              fetch(pe[r][c].price[w].w),
                              fetch(pe[r][c].price[s].s)));
            quantity.s=store(models(r,c,fetch(pe[r][c].price[n].n),
                              fetch(pe[r][c].price[e].e),
                              fetch(pe[r][c].price[w].w),
                              fetch(pe[r][c].price[s].s)));
        outlay.n=
            store(
                multiply(fetch(pe[r-1][c].exchangerate),
                multiply(     fetch(pe[r][c].price[n].n),
                              fetch(quantity.n)
                    )
                )
            );
        outlay.e=
            store(
                multiply(fetch(pe[r][c-1].exchangerate),
                multiply(     fetch(pe[r][c].price[e].e),
                              fetch(quantity.e)
                    )
                )
            );
        outlay.w=
            store(
                multiply(fetch(pe[r][c+1].exchangerate),
                multiply(     fetch(pe[r][c].price[w].w),
                              fetch(quantity.w)
                    )
                )
            );
        outlay.s=
            store(
                multiply(     fetch(pe[r+1][c].exchangerate),
                multiply(     fetch(pe[r][c].price[s].s),
                              fetch(quantity.s)
                    )
                )
            );
    pe[r][c].profit[n][e][w][s]=store(
                    add(fetch(outlay.n),
                    add     (
                              fetch(outlay.e),
                              add      (fetch(outlay.w),
                                        fetch(outlay.s)
                                        )
                              )
                    )
                    );
            }
        }
        }
        }
}


computeresiduals(r,c)
int r,c;
{
if (r>1)
                        pe[r][c].residual.n=fabs(pe[r][c].i.n-
                                pe[r-1][c].i.s
                        );
```

```
else
pe[r][c].residual.n=0;

if (c<COLS) pe[r][c].residual.e=fabs(pe[r][c].i.e-
                                     pe[r][c+1].i.w
                              );
else
pe[r][c].residual.e=0;
}


receivefrompartner(r,c)
int r,c;
{
if (r==1)
{
        pe[r][c].j.n=store(fetch(pe[r][c].i.n));
        pe[r][c].jp.n=store(fetch(pe[r][c].ip.n));
}
else
{
        pe[r][c].j.n=store(fetch(pe[r-1][c].i.s));
        pe[r][c].jp.n=store(fetch(pe[r-1][c].ip.s));
}
if (c==COLS)
        {
        pe[r][c].j.e=store(fetch(pe[r][c].i.e));
        pe[r][c].jp.e=store(fetch(pe[r][c].ip.e));
        }
else
        {
        pe[r][c].j.e=store(fetch(pe[r][c+1].i.w));
        pe[r][c].jp.e=store(fetch(pe[r][c+1].ip.w));
        }
}


calculatenewnodenorth(r,c)
int r,c;
{
extern double deltavoltage;
extern double vss;
double aa,bb,cc,dd;
 bb=    store(  divide  (
                        subtract(fetch(pe[r][c].ip.n),
                                fetch(pe[r][c].i.n)),
                        fetch(deltavoltage)
                        )
        );
 aa=    store(
        subtract(       fetch(pe[r][c].i.n),
                        multiply(
                                fetch(bb),
                                fetch(pe[r][c].v.n)
                                )
                )
        );
 dd=store(
                divide(
                subtract(fetch(pe[r][c].jp.n),
                        fetch(pe[r][c].j.n)
                        ),
                fetch(deltavoltage)
```

```
          )
          );
 cc=store(
          subtract        (fetch(pe[r][c].j.n),
                    multiply(fetch(dd),
                             fetch(pe[r][c].v.n)
                             )
                    )
          );

 if (absolute(subtract(fetch(bb),fetch(dd)))>fetch(0.00001))
   pe[r][c].new.n=store(divide(subtract(fetch(cc),fetch(aa)),
                               subtract(fetch(bb),fetch(dd))
                               )
                               );
 else
   pe[r][c].new.n=store(fetch(pe[r][c].v.n));
 if (fetch(pe[r][c].new.n)<fetch(-vss)) pe[r][c].new.n=store(fetch(-vss));
 if (fetch(pe[r][c].new.n)>fetch(vss)) pe[r][c].new.n=store(fetch(vss));

}
```

```
calculatenewnodeeast(r,c)
int r,c;
{
extern double deltavoltage;
extern double vss;
double aa,bb,cc,dd;
  bb=store(
          divide(subtract(fetch(pe[r][c].ip.e),
                          fetch(pe[r][c].i.e)
                         ),
             fetch(deltavoltage)
            )
          );
  aa=store(
          subtract(fetch(pe[r][c].i.e),
               multiply(fetch(bb),
                          fetch(pe[r][c].v.e)
                          )
               )
          );
 dd=store(
          divide(subtract(fetch(pe[r][c].jp.e),
                          fetch(pe[r][c].j.e)
                         ),
             fetch(deltavoltage)
            )
          );
 cc=store(
          subtract(fetch(pe[r][c].j.e),
               multiply(fetch(dd),
                          fetch(pe[r][c].v.e)
                          )
               )
          );
 if (absolute(subtract(bb,dd))>0.00001) pe[r][c].new.e=divide(subtract(cc,
                                                                aa
                                                               ),
                        subtract(bb,
                                 dd
                                )
```

```
                      );
  else pe[r][c].new.e=pe[r][c].v.e;
  if (pe[r][c].new.e<-vss) pe[r][c].new.e=-vss;
  if (pe[r][c].new.e>vss) pe[r][c].new.e=vss;
}
```

calculatenewnodes(r,c)

```
int r,c;
{
  calculatenewnodenorth(r,c);
  calculatenewnodeeast(r,c);
}
```

updatenewnodes(r,c)

```
int r,c;
{
        pe[r][c].price[2].e=pe[r][c].new.e;
        pe[r][c].price[2].n=pe[r][c].new.n;
        pe[r-1][c].price[2].s=pe[r][c].new.n;
        pe[r][c+1].price[2].w=pe[r][c].new.e;

        pe[r][c].v.e=pe[r][c].new.e;
        pe[r][c].v.n=pe[r][c].new.n;
        pe[r-1][c].v.s=pe[r][c].new.n;
        pe[r][c+1].v.w=pe[r][c].new.e;
}
```

saveedgevoltages(r,c)

```
int r,c;
{
  pe[r][c].v0.n=pe[r][c].v.n;
  pe[r][c].v0.e=pe[r][c].v.e;
  pe[r][c].v0.w=pe[r][c].v.w;
  pe[r][c].v0.s=pe[r][c].v.s;

 pe[r][c].price[1].n=pe[r][c].v0.n;
 pe[r][c].price[1].e=pe[r][c].v0.e;
 pe[r][c].price[1].w=pe[r][c].v0.w;
 pe[r][c].price[1].s=pe[r][c].v0.s;
}
```

saveedgecurrents(r,c)

```
int r,c;
{
        pe[r][c].i0.n=pe[r][c].i.n;
        pe[r][c].i0.e=pe[r][c].i.e;
        pe[r][c].i0.w=pe[r][c].i.w;
        pe[r][c].i0.s=pe[r][c].i.s;
}
```

updateedgevoltages(r,c)

```
int r,c;
{
  if (c>1)              pe[r][c].v0.w=pe[r][c].v.w;
  if (c<COLS)     pe[r][c].v0.e=pe[r][c].v.e;
  if (r>1)              pe[r][c].v0.n=pe[r][c].v.n;
  if (r<ROWS)           pe[r][c].v0.s=pe[r][c].v.s;
}
```

```
solvefornewnodes(r,c)
int r,c;
{
  computecurrents(r,c);
  receivefrompartner(r,c);
  calculatenewnodes(r,c);
  updatenewnodes(r,c);
}


iterate()
{
extern int iterations;
int r,c;
  for (r=1;r<=ROWS;r=r++) {for (c=1;c<=COLS;c=c+1)
    {saveedgevoltages(r,c);}};
  for (r=1;r<=ROWS;r=r++) {for (c=1;c<=COLS;c=c+1)
    {saveedgecurrents(r,c);}};
  for (r=1;r<=ROWS;r=r++) {for (c=1;c<=COLS;c=c+1)
   {computecurrents(r,c);}};
  for (r=1;r<=ROWS;r=r++) {for (c=1;c<=COLS;c=c+1)
   {computeresiduals(r,c);}};
  for (r=1;r<=ROWS;r=r++) {for (c=1;c<=COLS;c=c+1)
   {solvefornewnodes(r,c);}};
  for (r=1;r<=ROWS;r=r++) {for (c=1;c<=COLS;c=c+1)
   {updateedgevoltages(r,c);}};
  if (applygamecriterion[iterations]==TRUE)
  {
          for (r=1;r<=ROWS;r=r++)
          {
          for (c=1;c<=COLS;c=c++)
           computepossibles(r,c);
          }
           for (r=1;r<=ROWS;r=r++)
           {
           for (c=1;c<=COLS;c=c++)
            maxprofitcriterion(r,c);
           }
  }
}


int  alldone()
{int r,c;
 int flag;
  for (r=1;r<=ROWS;r++)
  {
    for (c=1;c<=COLS;c++)
     {
          if ( (pe[r][c].residual.n<currentlimit)
           && (pe[r][c].residual.e<currentlimit) )
       pe[r][c].done=TRUE;
          else
          pe[r][c].done=FALSE;
     }
  }
          flag=TRUE;
          for (r=1;r<=ROWS; r++)
          {for (c=1;c<=COLS; c++)
          {if (pe[r][c].done==FALSE) flag=FALSE; }}
   return flag;
}
```

```
computecapacitorincrement()
{
extern double dt;
int r,c;
double dqn,dqe,dqw,dqs,dq;
for (r=1;r<=ROWS;r++)
{
        for (c=1;c<=COLS;c++)
        {
                if (pe[r][c].cap.c>0)
                {
                        dqn=pe[r][c].cap.jn*dt;
                        dqe=pe[r][c].cap.je*dt;
                        dqw=pe[r][c].cap.jw*dt;
                        dqs=pe[r][c].cap.js*dt;
                        dq=dqn+dqe+dqw+dqs;
                        pe[r][c].cap.dv=-dq/pe[r][c].cap.c;
                }
        else
                pe[r][c].cap.dv=0;
        }
}
}


double  largestresidual()
{
double largest;
int r,c;

largest=0;
for (r=1;r<=ROWS;r++)
{
        for (c=1;c<=COLS;c++)
        {
                if (fabs(pe[r][c].residual.n)>largest)
                        largest=fabs(pe[r][c].residual.n);
                if (fabs(pe[r][c].residual.e)>largest)
                        largest=fabs(pe[r][c].residual.e);
        }
}
return largest;
}


savenodes()
{
extern FILE *outputfile;
int r,c;
        outputfile=fopen("g.end","w");
        if (outputfile==NULL) printf("G.END WILL NOT OPEN\n");
        else printf("final nodes saved in readable text form in G.END\n");
        for (r=1;r<=ROWS;r++)
        {
        for (c=1; c<=COLS; c++)
                fprintf(outputfile,"%15.12f",pe[r][c].v.n);
        fprintf(outputfile,"\n");

        for (c=1; c<=COLS; c++)
                fprintf(outputfile,"%15.12f",pe[r][c].v.e);
        fprintf(outputfile,"\n");

        for (c=1; c<=COLS; c++)
```

```
            fprintf(outputfile,"%15.12f",pe[r][c].v.w);
        fprintf(outputfile,"\n");

        for (c=1; c<=COLS; c++)
            fprintf(outputfile,"%15.12f",pe[r][c].v.s);
        fprintf(outputfile,"\n");
        }
}
```

savenodesininputformat()

```
{
int r,c;
int dummy;
FILE *nodefile;

printf("voltage nodes saved in input format to G.NODES.INPUT\n");
nodefile=fopen("g.nodes.input","w");

for (r=1;r<=ROWS;r++)
{
 for (c=1;c<=COLS;c++) fprintf(nodefile,"%12.9lf",pe[r][c].v.n);
 fprintf(nodefile,"\n");
 for (c=1;c<=(COLS+1);c++)  fprintf(nodefile,"%12.9lf",pe[r][c].v.w);
 fprintf(nodefile,"\n");
}
 for (c=1;c<=COLS;c++)
 fprintf(nodefile,"%12.9lf",pe[ROWS+1][c].v.n);
  fprintf(nodefile,"\n");

 for (r=1;r<=ROWS;r++)
 {
 fprintf(nodefile,"    ");
 for (c=1;c<=COLS;c++)
 fprintf(nodefile,"%12.9lf",pe[r][c].cap.v);
  fprintf(nodefile,"\n");
 }
}
```

reportprogress()

```
{
extern int iterations;
extern int progress;

  printf("%4d iterations out of %d    max. residual=%9.0f\n",
  iterations,granditerations,(1000000.0*largestresidual()));
  progress=0;
}
```

```
double   expf(x)
double x;
{
  double r;
  double sum;
  double n;
  int c,count;
  if (masking==FALSE) return exp(x);
  count=0;        /* if x ok*/
  if (x>1.0)
  {
     for (c=0; x>1.0; c++) /* cut x down, keep count */
     {
      x=x/2;
     }
   count=c;
}
   /* when get here, count=number of times halved*/

 /* calculate exponent of number <=1 */
  r=1.0;
  n=1.0;
  sum=1.0;
  for (n=1.0; n<15.0; n=n+1)
  {
   r=-multiply(x,
                divide(r,
                       n
                      )
               );
   sum=add(sum,
            r
           );
  }
/* if it was halved (sqrt) then power back up count times */
   if (count>0)
   {for (c=1; c<=count; c=c++)
    sum=multiply(sum,
                  sum
                 );
    }
   return divide(1.0,
                  sum
                 );
}


double   multiply(a,b)
double a,b;
{
double c;
if (masking==FALSE) return (a*b);
mask(&a);
mask(&b);
c=a*b;
mask(&c);
numbermultiply++;
return c;
}
```

```
double  divide(a,b)
double a,b;
{
double c;
if (masking==FALSE) return (a/b);
mask(&a);
mask(&b);
c=a/b;
mask(&c);
numberdivide++;
return c;
}


double  add(a,b)
double a,b;
{
double c;
if (masking==FALSE) return (a+b);
mask(&a);
mask(&b);
c=a+b;
mask(&c);
numberadd++;
return c;
}


double  subtract(a,b)
double a,b;
{
double c;
if (masking==FALSE) return (a-b);
mask(&a);
mask(&b);
c=a-b;
mask(&c);
numbersubtract++;
return c;
}


double  fetch(a)
double a;
{
if (masking==FALSE) return a;
mask(&a);
numberfetch++;
return a;
}


double  store(a)
double a;
{
if (masking==FALSE) return a;
mask(&a);
numberstore++;
return a;
}
```

```
double   absolute(a)
double a;
{
if (masking==FALSE) return fabs(a);
mask(&a);
numberabsolute++;
return fabs(a);
}


mask(s)
/* mask a double float number*/
/* s=address of double */

unsigned char *s;
{
unsigned char *m;

m=&bitmask[0];
*s=*s & *m; s++; m++;
*s=*s & *m; s++; m++;
*s=*s & *m; s++; m++;
*s=*s & *m; s++; m++;
*s=*s & *m; s++; m++;
*s=*s & *m; s++; m++;
*s=*s & *m; s++; m++;
*s=*s & *m;
}


crlf()
{
        printf("\n");
}
```

```
setupbitmask(bits)
int bits;
{
int i;
switch (bits)
{
case  1: for (i=0;i<8;i++) bitmask[i]=maskset1[i]; return;
case  2: for (i=0;i<8;i++) bitmask[i]=maskset2[i]; return;
case  3: for (i=0;i<8;i++) bitmask[i]=maskset3[i]; return;
case  4: for (i=0;i<8;i++) bitmask[i]=maskset4[i]; return;
case  5: for (i=0;i<8;i++) bitmask[i]=maskset5[i]; return;
case  6: for (i=0;i<8;i++) bitmask[i]=maskset6[i]; return;
case  7: for (i=0;i<8;i++) bitmask[i]=maskset7[i]; return;
case  8: for (i=0;i<8;i++) bitmask[i]=maskset8[i]; return;
case  9: for (i=0;i<8;i++) bitmask[i]=maskset9[i]; return;
case 10: for (i=0;i<8;i++) bitmask[i]=maskset10[i]; return;
case 11: for (i=0;i<8;i++) bitmask[i]=maskset11[i]; return;
case 12: for (i=0;i<8;i++) bitmask[i]=maskset12[i]; return;
case 13: for (i=0;i<8;i++) bitmask[i]=maskset13[i]; return;
case 14: for (i=0;i<8;i++) bitmask[i]=maskset14[i]; return;
case 15: for (i=0;i<8;i++) bitmask[i]=maskset15[i]; return;
case 16: for (i=0;i<8;i++) bitmask[i]=maskset16[i]; return;
case 17: for (i=0;i<8;i++) bitmask[i]=maskset17[i]; return;
case 18: for (i=0;i<8;i++) bitmask[i]=maskset18[i]; return;
case 19: for (i=0;i<8;i++) bitmask[i]=maskset19[i]; return;
case 20: for (i=0;i<8;i++) bitmask[i]=maskset20[i]; return;
case 21: for (i=0;i<8;i++) bitmask[i]=maskset21[i]; return;
case 22: for (i=0;i<8;i++) bitmask[i]=maskset22[i]; return;
case 23: for (i=0;i<8;i++) bitmask[i]=maskset23[i]; return;
case 24: for (i=0;i<8;i++) bitmask[i]=maskset24[i]; return;
case 25: for (i=0;i<8;i++) bitmask[i]=maskset25[i]; return;
case 26: for (i=0;i<8;i++) bitmask[i]=maskset26[i]; return;
case 27: for (i=0;i<8;i++) bitmask[i]=maskset27[i]; return;
case 28: for (i=0;i<8;i++) bitmask[i]=maskset28[i]; return;
case 29: for (i=0;i<8;i++) bitmask[i]=maskset29[i]; return;
case 30: for (i=0;i<8;i++) bitmask[i]=maskset30[i]; return;
case 31: for (i=0;i<8;i++) bitmask[i]=maskset31[i]; return;
case 32: for (i=0;i<8;i++) bitmask[i]=maskset32[i]; return;
case 33: for (i=0;i<8;i++) bitmask[i]=maskset33[i]; return;
case 34: for (i=0;i<8;i++) bitmask[i]=maskset34[i]; return;
case 35: for (i=0;i<8;i++) bitmask[i]=maskset35[i]; return;
case 36: for (i=0;i<8;i++) bitmask[i]=maskset36[i]; return;
case 37: for (i=0;i<8;i++) bitmask[i]=maskset37[i]; return;
case 38: for (i=0;i<8;i++) bitmask[i]=maskset38[i]; return;
case 39: for (i=0;i<8;i++) bitmask[i]=maskset39[i]; return;
case 40: for (i=0;i<8;i++) bitmask[i]=maskset40[i]; return;
case 41: for (i=0;i<8;i++) bitmask[i]=maskset41[i]; return;
case 42: for (i=0;i<8;i++) bitmask[i]=maskset42[i]; return;
case 43: for (i=0;i<8;i++) bitmask[i]=maskset43[i]; return;
case 44: for (i=0;i<8;i++) bitmask[i]=maskset44[i]; return;
case 45: for (i=0;i<8;i++) bitmask[i]=maskset45[i]; return;
case 46: for (i=0;i<8;i++) bitmask[i]=maskset46[i]; return;
case 47: for (i=0;i<8;i++) bitmask[i]=maskset47[i]; return;
case 48: for (i=0;i<8;i++) bitmask[i]=maskset48[i]; return;
case 49: for (i=0;i<8;i++) bitmask[i]=maskset49[i]; return;
case 50: for (i=0;i<8;i++) bitmask[i]=maskset50[i]; return;
case 51: for (i=0;i<8;i++) bitmask[i]=maskset51[i]; return;
case 52: for (i=0;i<8;i++) bitmask[i]=maskset52[i]; return;
case 53: for (i=0;i<8;i++) bitmask[i]=maskset53[i]; return;
}
}
```

Appendix F

SPICE deck Listing

```
MULTIPLIER CELL PROGRAM  - MEAN VALUES OF PARAMETERS
.MODEL ENH  NMOS       VTO= 0.850 GAMMA= 0.620 PHI=0.65 LAMBDA=0.06
+               LEVEL=2
+               CGSO=4E-10  CGDO=4E-10   CGBO=2E-10
+               CBD=20FF    CBS=20FF
+               RS=1.0      RD=1.0
+               TOX=1E-7    NSUB=2E15
.MODEL DEPL NMOS       VTO=-2.750 GAMMA= 0.650 PHI=0.65 LAMBDA=0.04
+               LEVEL=2
+               CGSO=4E-10  CGDO=4E-10    CGBO=2E-10
+               CBD=20FF    CBS=20FF
+               RS=1.0      RD=1.0
+             TOX=1E-7     NSUB=2E15
.SUBCKT YINV1  1 2 3
M1 3 2 2 0 DEPL     L=22.5U W= 5.0U  AD=112.0P AS=100.0P
M2 2 1 0 0 ENH      L= 5.0U   W=10.0U  AD=100.0P AS=125.0P
.ENDS  YINV1
.SUBCKT YINV2  1 2 3
M3 3 2 2 0 DEPL     L=20.0U   W= 5.0U   AD=112.0P AS=137.0P
M4 2 1 0 0 ENH      L= 5.0U   W= 5.0U   AD=137.0P AS=187.0P
.ENDS  YINV2
.SUBCKT XINV1  1 2 3
M5 3 2 2 0 DEPL     L=22.5U W= 5.0U  AD=112.0P AS=100.0P
M6 2 1 0 0 ENH      L= 5.0U   W=10.0U  AD=100.0P AS=125.0P
.ENDS  XINV1
.SUBCKT XINV2  1 2 3
M7 3 2 2 0 DEPL     L=20.0U   W= 5.0U   AD=112.0P AS=100.0P
M8 2 1 0 0 ENH      L= 5.0U   W= 5.0U   AD=100.0P AS=125.0P
.ENDS  XINV2
.SUBCKT AINV1  1 2 3
M9  3 2 2 0 DEPL    L=20.0U   W= 5.0U   AD=112.0P AS=100.0P
M10 2 1 0 0 ENH     L= 5.0U   W=10.0U  AD=100.0P AS=125.0P
.ENDS  AINV1
.SUBCKT AINV2  1 2 3
M11 3 2 2 0 DEPL    L=20.0U   W= 5.0U   AD=112.0P AS= 87.0P
M12 2 1 0 0 ENH     L= 5.0U   W= 5.0U   AD= 87.0P  AS=112.0P
.ENDS  AINV2
.SUBCKT CINV1  1 2 3
M13 3 2 2 0 DEPL    L=22.5U W= 5.0U  AD=112.0P AS=100.0P
M14 2 1 0 0 ENH     L= 5.0U   W=10.0U  AD=100.0P AS=125.0P
.ENDS  CINV1
.SUBCKT CINV2  1 2 3
M15 3 2 2 0 DEPL    L=20.0U   W= 5.0U   AD=112.0P AS=137.0P
M16 2 1 0 0 ENH     L= 5.0U   W= 5.0U   AD=137.0P AS=112.0P
.ENDS  CINV2
.SUBCKT INV    1 2 3
M17 3 2 2 0 DEPL    L=20.0U   W= 5.0U   AD=112.0P AS= 87.0P
M18 2 1 0 0 ENH     L= 5.0U   W= 5.0U   AD= 87.0P  AS=125.0P
.ENDS  INV
.SUBCKT NAND   1 2 3 4
M19 4 3 3 0 DEPL    L=40.0U   W= 5.0U   AD=250.0P AS= 20.0P
M20 3 1 5 0 ENH     L= 5.0U   W=10.0U  AD=250.0P AS= 75.0P
M21 5 2 0 0 ENH     L= 5.0U   W=10.0U  AD= 75.0P  AS=123.0P
.ENDS  NAND
.SUBCKT MUX 1 2 3 4 5 6 7 8 9
M1     81 2 82 0   ENH  L= 5.0U  W= 5.0U  AS= 60.0P  AD= 60.0P
M2     82 4 84 0   ENH  L= 5.0U  W= 5.0U  AS= 60.0P  AD= 60.0P
M3     85 10 86 0  ENH  L= 5.0U  W= 5.0U  AS= 60.0P  AD= 60.0P
M4     86 31 88 0  ENH  L= 5.0U  W= 5.0U  AS= 60.0P  AD= 60.0P
M5     89 21 90 0  ENH  L= 5.0U  W= 5.0U  AS= 60.0P  AD= 60.0P
M6     90 31 92 0  ENH  L= 5.0U  W= 5.0U  AS= 60.0P  AD= 60.0P
M7     93 11 94 0  ENH  L= 5.0U  W= 5.0U  AS= 60.0P  AD= 60.0P
M8     94 42 96 0  ENH  L= 5.0U  W= 5.0U  AS= 60.0P  AD= 60.0P
M9     63 22 62 0  ENH  L= 5.0U  W= 5.0U  AS= 60.0P  AD= 60.0P
M10     62 42 60 0  ENH  L= 5.0U  W= 5.0U  AS= 60.0P  AD= 60.0P
```

```
M11    66 12 65 0  ENH  L= 5.0U  W= 5.0U  AS= 60.0P  AD= 60.0P
M12    65 33 64 0  ENH  L= 5.0U  W= 5.0U  AS= 60.0P  AD= 60.0P
M13    70 23 69 0  ENH  L= 5.0U  W= 5.0U  AS= 60.0P  AD= 60.0P
M14    69 33 67 0  ENH  L= 5.0U  W= 5.0U  AS= 60.0P  AD= 60.0P
M15    74 13 73 0  ENH  L= 5.0U  W= 5.0U  AS= 60.0P  AD= 60.0P
M16    73 44 71 0  ENH  L= 5.0U  W= 5.0U  AS= 60.0P  AD= 60.0P
*METAL MAINLY
C1  1  0  0.005000P
C10 10 0  0.005000P
C11 11 0  0.005000P
C12 12 0  0.005000P
C13 13 0  0.005000P
C2  2  0  0.005000P
C21 21 0  0.005000P
C22 22 0  0.005000P
C23 23 0  0.005000P
C24 24 0  0.005000P
C3  3  0  0.005000P
C31 31 0  0.005000P
C32 32 0  0.005000P
C33 33 0  0.005000P
C34 34 0  0.005000P
C4  4  0  0.005000P
C41 41 0  0.005000P
C42 42 0  0.005000P
C43 43 0  0.005000P
C44 44 0  0.005000P
C50 5  0  0.010000P
*MAINLY DIFFUSION
R1  1  10      1.000
R10 10 11      1.000
R11 11 12      1.000
R12 12 13      1.000
R2  2  21      1.000
R21 21 22      1.000
R22 22 23      1.000
R23 23 24      1.000
R3  3  31      1.000
R31 31 32      1.000
R32 32 33      1.000
R33 33 34      1.000
R4  4  41      1.000
R41 41 42      1.000
R42 42 43      1.000
R43 43 44      1.000
R84 84 0       1.000
R88 88 5       1.000
R92 92 6       1.000
R96 96 6       1.000
R60 60 6       1.000
R64 64 6       1.000
R67 67 7       1.000
R71 71 7       1.000
R81 81 8       1.000
R85 85 8       1.000
R89 89 8       1.000
R93 93 8       1.000
R66 66 9       1.000
R70 70 9       1.000
R74 74 9       1.000
R63 63 9       1.000
.ENDS MUX
*NOMINAL CIRCUIT
*DIFFUSION
R16     21 31   600.00
```

```
R17    23 30   580.00
*METAL
R3     5 6    850.00
*POLYSILICON
R1     1 2    343.0
R2     3 4    223.0
R4     8 24    247.0
R5     9 10    285.0
R6     11 12   177.0
R7     13 7     80.0
R8     8 27    353.0
R9     25 26   187.0
R10     14 15   587.0
R11     16 17   260.0
R12     18 28   180.0
R13     16 29   347.0
R14     19 20   357.0
R15     21 22   170.0
*CAPACITORS
*DIFF
C17    23 0    0.070000P
*METAL
C3     5 0     0.200P
C5     9 0     0.050P
C7     13 0    0.004P
C9     25 0    0.009P
*POLY
C132    132 0   0.010P
C133    133 0   0.010P
C4     8 0    0.120P
C1     1 0    0.100P
C2     3 0    0.010P
C6     11 0    0.008P
C8     8 0    0.030P
C10     14 0    0.050P
C11     16 0    0.010P
C12     18 0    0.009P
C13     16 0    0.020P
C14     19 0    0.080P
C15     21 0    0.009P
C16     21 0    0.099P
*GATES
X1      2  3  99  YINV1
X2      4  5  99  YINV2
X3     10 11  99  XINV1
X4     12 13  99  XINV2
X5     15 16  99  AINV1
X6     17 18  99  AINV2
X7     20 21  99  CINV1
X8     22 23  99  CINV2
X9      6 7 8  99  NAND
X10     24 25  99  INV
X11     26 27 28 29 99 30 31 132 133  MUX
*VOLTAGES
VDD    99 0 DC 5.0
VCIN   19 0   0  PULSE(0 5 1N 5N 5N 200N 400N)
VSIN   14 0   0  PULSE(0 5 1N 5N 5N 200N 400N)
VXIN   9 0    0  PULSE(0 5 1N 5N 5N 200N 400N)
VYIN   1 0    0  PULSE(0 5 1N 5N 5N 200N 400N)
*SIMULATION
.PRINT TRAN      V(132) V(133)
.OPTIONS  TNOM=27.0   NOPAGE
.TRAN  1NSEC 150NSEC
.END
```

Bibliography

[A$^+$88] A.R. Alvarez et al. Application of Statistical Design and Response Surface Methods to Computer-aided VLSI Device Design. *IEEE Transactions on Computer-Aided Design*, CAD-6:272–288, February 1988.

[ABHS89] M.C. August, G.M. Brost, C.C. Hsiung, and A.J. Schliffleger. Cray X-MP: The Birth of a Supercomputer. *Computer*, 22(1):45–52, January 1989.

[AC89] B.D. Ackland and R.A. Clark. Event-EMU: An Event Driven Timing Simulator for MOS VLSI Circuits. *Proceedings of the International Conference on Computer-Aided Design*, pages 80–83, November 1989.

[ACS85] V. Ashok, R. Costello, and P. Sadayappan. Distributed Discrete Event Simulation using Dataflow. In *Proceedings of the International Conference on Parallel Processing*, pages 503–510, August 1985.

[AG75] R.B. Ash and M.F. Gardner. *Topics in Stochastic Processes.* Academic Press, New York, 1975.

[AG82] B.W. Arden and R. Ginosar. MP/C: A Multiprocessor/Computer Architecture. *IEEE Transactions on Computers*, C-31(5):455–473, May 1982.

[AJ88] R. Agrawal and H.V. Jagadish. Partitioning Techniques for Large-Grained Parallelism. *IEEE Transactions on Computers*, 37(12):1627–1634, December 1988.

[Amb85] A. Ambler. Microprocessor Design Speeds Simulation. *Electronics Week*, pages 30–31, April 1985.

[Arn74] L. Arnold. *Stochastic Differential Equations: Theory and Applications.* John Wiley, New York, 1974.

[Ban88] P. Banerjee. A Tutorial on the Use of Parallel Processing in VLSI CAD Applications: Parallel Algorithms for Design Verification. In *International Conference on Computer Aided Design*, November 1988.

[BD92] K. Bogineni and P.W. Dowd. An Optically Interconnected Distributed Shared Memory System: Architecture and Performance Analysis. *International Journal of High Speed Computing*, 4(3):179–212, 1992.

[Bel89] C.G. Bell. The Future of High Performance Computers in Science and Engineering. *Communications of the ACM*, 32(9):1091–1101, September 1989.

[BF90] N. Bergmann and X. Fan. Data Formats and Arithmetic Operators for Serial/Parallel Trade-Offs in Pipelined Architectures. *IEEE International Symposium on Circuits and Systems*, pages 1248–1251, May 1990.

[BHSV81] R.K. Brayton, G.D. Hachtel, and A.L. Sangiovanni-Vincentelli. A Survey of Optimization Techniques for Integrated-Circuit Design. *Proceedings of the IEEE*, 69(10), October 1981.

[BJS90] P. Banerjee, M.H. Jones, and J.S. Sargent. Parallel Simulated Annealing Algorithms for Cell Placement on Hypercube Multiprocessors. *IEEE Transactions on Parallel and Distributed Systems*, 1(1):91–106, January 1990.

[BKB82] D. Bondurant, M. Kopman, and P. Bytheway. A High-Performance Configurable Microprocessor. *VLSI Design*, pages 16–19, November 1982.

[Bra75] J.V. Bradley. *Probability; Decision; Statistics.* Prentice-Hall, Englewood Cliffs, N.J., 1975.

[Bro88] A. Brocco. Macromodeling CMOS Circuits. *IEEE Transactions on Computer-Aided Design*, 7(12):1240–1248, December 1988.

[Bry84] R.E. Bryant. A Switch-Level Model and Simulator for MOS Digital Systems. *IEEE Transactions on Computers*, C-33(2):160–177, February 1984.

[Bry88] R.E. Bryant. Data Parallel Switch-Level Simulation. In *Proceedings of the International Conference on Computer-Aided Design*, pages 354–357, November 1988.

[BS80] R.K. Brayton and R. Spence. *Sensitivity and Optimisation.* -, 1980.

[BS87] J. Benkoski and A.J. Strojwas. A New Approach to Heirarchical and Statistical Timing Simulations . *IEEE Transactions on Computer-Aided Design*, CAD-6(6):1039–1052, November 1987.

[BT92] I. Buch and Y. Taumann. Bargaining with a Ruler. *International Journal of Game Theory*, 21:131–148, 1992.

[cai90] In E.R. Caianiello, editor, *Third Italian Workshop on Parallel Architectures and Neural Networks*, page 65, Singapore, May 1990. World Scientific.

[CBE86] P. Cox, R. Burch, and B. Epler. Circuit Partitioning for Parallel Processing. In *Proceedings of the International Conference on Computer-aided Design*, pages 186–189, November 1986.

[CFL91] V. Cantoni, M. Ferretti, and L. Lombardi. A Comparison of Homogeneous Hierarchical Interconnection Structures. *Proceedings of the IEEE*, 79(4):416–428, April 1991.

[CG86] A.E. Charlesworth and J.L. Gustafson. Introducing Replicated VLSI to Supercomputing: the FPS-164/MAX Scientific Computer. *Computer*, pages 11–23, March 1986.

[CGR83] W.M. Coughran, Jr., E. Grosse, and D.J. Rose. CAzM: A Circuit Analyzer with Macromodelling. *IEEE Transactions on Electron Devices*, ED-30(9):1207–1209, September 1983.

[Chi69] Paul M. Chirlian. *Basic Network Theory.* McGraw-Hill, 1969.

[CMS92] W.E. Clark, G.L. McColm, and W.R. Stark. On the Complexity of Deadlock-Free Programs on a Ring of Processors. *Journal of Parallel and Distributed Computing*, 16:67–71, 1992.

[Coh83] P.B. Cohen. Layout Considerations in Predicting VLSI Performance. *VLSI Design*, pages 64–65, January 1983.

[Col82] A. Colman. *Game Theory and Experimental Games: The Study of Strategic Interaction.* Pergamon Press, 1982.

[CSV88] G. Casinovi and A. Sangiovani-Vincentelli. A New Aggregation Technique for the Solution of Large Systems of Algebraic Equations. *IEEE Transactions on Computer-Aided Design*, 7(9):976–986, September 1988.

[CT84] N.H. Christ and A.E. Terrano. A Very Fast Parallel Processor. *IEEE Transactions on Computers*, 33(4), April 1984.

[CWT83] M. Canepa, E. Weber, and H. Talley. VLSI in Focus: Designing a 32-bit CPU Chip. *VLSI Design*, pages 20–24, January 1983.

[CYC84] P. Cox, P. Yang, and P. Chatterjee. Statistical Modelling for Efficient Parametric Yield Estimation. In *Proceedings of the International Electron Device Meeting*, pages 242–245, 1984.

[DCR82] T. Downs, A.S. Cook, and P.G. Rogers. A Partitioning Approach to the Statistical Design of Large Circuits and Systems. *International Symposium on Circuits and Systems*, 1:138, 1982.

[Det86] R. Dettmer. The Artful Transputer. *Electronics and Power*, August 1986.

[DGT85] J. Demongeot, E. Goles, and M. Tchuente, editors. *Dynamical Systems and Cellula Automata.* Academic Press, Paris, 1985.

[DKP83] M. Denneau, E. Kronstadt, and G. Pfister. Design and Implementation of a Software Simulation Engine. *Computer Aided Design*, 15(3), May 1983.

[DLS86] J.T. Deutsch, T.D. Lovett, and M.L. Squires. Parallel Computing for VLSI Circuit Simulation. *VLSI Systems Design*, pages 46–52, July 1986.

[DMS88] S.W. Director, W. Maly, and A.J. Strojwas. *VLSI Design for Manufacturing: Yield Enhancement.* Kluwer Academic Publishers, Boston, 1988.

[DOCM87] D. Dumlugol, P. Odent, J.P. Cockx, and H.J.D. Man. Switch-Electrical Segmented Waveform Relaxation for Digital MOS VLSI and its Acceleration on Parallel Computers. *IEEE Transactions on Computer-Aided Design*, CAD-6(6):992–1005, November 1987.

[Don88] J.J. Dongarra. Experimental Parallel Computing Architectures. In *Special Topics in Supercomputing*, Amsterdam, 1988. Elsevier Science.

[DSSSG91] F. Distante, M. Sami, R. Stefanelli, and G. Storti-Gajani. Mapping Neural Nets onto a Massively Parallel Architecture: A Defect-Tolerance Solution. *Proceedings of the IEEE*, 79(4):444–460, April 1991.

[Dun90] R. Duncan. A Survey of Parallel Computer Architectures. *Computer*, 23(2):5–16, February 1990.

[EAKHAK92] M.O. Esonu, A.J. Al-Khalili, S. Harir, and D. Al-Khalili. Systolic Arrays: How to Choose Them. In *IEE Proceedings-E*, volume 139, pages 179–188, May 1992.

[Eic92] J. Eichberger. Cooperative Game Theory; Games with Incomplete Information; Bargaining Theory; Zero-Sum Games. *In Research Papers: University of Melbourne, Victoria, Australia*, (284, 290, 330, 332), 1991-1992.

[FAG88] D.K. Ferry, L.A. Akers, and E.W. Greeneich. *Ultra Large Scale Integrated Microelectronics.* Prentice Hall, New York, 1988.

[FHJ+83] H. Fromm, U. Hercksen, K-H. John, R. Klar, and W. Kleindoder. Experiences with Performance Measurement and Modeling of a Processor Array. *IEE Transactions on Computers*, C-32(1):15–19, January 1983.

[Fre75] S.L. Freeney. Special Purpose Hardware for Digital Filtering. *Proceedings of the I.E.E.E.*, April 1975.

[Gab86] R.P. Gabriel. Massively Parallel Computers: The Connection Machine and NON-VON. *Science*, 231:975–978, February 1986.

[GGK+83] A. Gottlieb, R. Grishman, C.P. Kruskal, K.P. McAuliffe, L. Rudolf, and M. Snir. The NYU Ultracomputer - Designing an MIMD Shared Memory Parallel Computer. *IEEE Transactions on Computers*, C-32(2):175–189, February 1983.

[GJMW91] K. Gallivan, W. Jalby, A. Malony, and H. Wijshoff. Performance Prediction for Parallel Numerical Algorithms. *International Journal of High Speed Computing*, 3(1):31–62, 1991.

[GS79] I.I. Gilman and A.V. Skorohod. *The Theory of Stochastic Processes, Volume III.* Springer-Verlag, Berlin, 1979.

[GS82] S. Garcia and K.S. Sriram. A Survey of IC CAD Tools for Design, Layout and Testing. In *VLSI DESIGN*, pages 68–73, September 1982.

[Hag81] L.A. Hageman. *Applied Iterative Methods.* Academic Press, New York, 1981.

[HB86] N. Herr and J.J. Barnes. Statistical Circuit Simulation modelling of CMOS VLSI. *IEEE Transactions on Computer-Aided Design*, CAD-5:15–22, January 1986.

[Hil82] W.D. Hillis. New Computer Architectures and their Relationship to Physics or Why Computer Science is No Good. *International Journal of Theoretical Physics*, 21(3/4):255–262, 1982.

[Hit82] R.B. Hitchcock. Timing Verification and the Timing Analysis Program. In *19th Design Automation Conference*, page 594, 1982.

[HM89] J.P. Hayes and T. Mudge. Hypercube Supercomputers. *Proceedings of the IEEE*, 77(12):1829–1841, December 1989.

[HNSB90] D.S. Harrison, A.R. Newton, R.L. Spickelmier, and T.J. Barnes. Electronic CAD Frameworks. *Proceedings of the IEEE*, 78(2):393–417, February 1990.

[Hon87] R.W. Hon. Dynamic Analysis Tools. In S.M. Rubin, editor, *Computer Aids for VLSI Design*, Reading, Mass., 1987. Addison-Wesley.

[HP90] J.L. Hennessy and D.A. Patterson. *Computer Architecture: A Quantitative Approach.* Morgan Kaufmann, San Mateo, CA, 1990.

[Ich83] T. Ichiishi. *Game Theory for Economic Analysis.* Academic Press, New York, 1983.

[Jaz70] A.H. Jazwinski. *Stochastic Process and Filtering Theory.* Academic Press, New York, 1970.

[Jou87] N.P. Jouppi. Timing Analysis and Performance Improvement of MOS VLSI Designs. *IEEE Transactions on Computer-Aided Design*, CAD-6(4):650–665, July 1987.

[KO90] E.S. Kuh and T. Ohtsuki. Recent Advances in VLSI Layout. In *Proceedings of the IEEE*, page 237, February 1990.

[Kob82] R.K. Koblitz. Interactive Design Centering by an Efficient Assessment Criterion. *IEEE Transactions on Circuits and Systems*, pages 130–133, 1982.

[Kun87] S. Y. Kung. *VLSI Array Processors.* Prentice Hall, 1987.

[Lan86] H.W. Lang. The Instruction Systolic Array, A Parallel Architecture for VLSI. *Integration VLSI Journal*, 4:65–74, 1986.

[Lew72] D. Lewin. *Theory and Design of Digital Computers.* Nelson, 1972.

[Lew88] D.M. Lewis. Hardware Accelerators for Timing Simulation of VLSI Digital Circuits. *IEEE Transactions on Computer-Aided Design*, 7(11):1134–1149, November 1988.

[Lig87] M.R. Lightner. Modeling and Simulation of VSLI Digital Systems. In *Proceedings of the IEEE*, volume 75, pages 786–796, 1987.

[LJ91] R.M. Lea and I.P. Jalowiecki. Associative Massively Parallel Computers. *Proceedings of the IEEE*, 79(4):469–479, April 1991.

[LMP82] Y.H. Levendel, P.R. Menon, and S.H. Patel. Special-Purpose Computer Logic Simulation using Distributed Processing. *The Bell System Technical Journal*, 61(10):2873–2909, December 1982.

[LR88] S.W. Lee and R.C. Rennick. A Compact IGFET Model - ASIM. *IEEE Transactions on Computer-Aided Design*, 7(9):952–975, September 1988.

[LS91] H. Li and Q.F. Stout. Reconfigurable SIMD Massively Parallel Computers. *Proceedings of the IEEE*, 79(4):429–443, April 1991.

[LSV82] E Lelarasmee and A.L. Sangiovanni-Vincentelli. A New Relaxation Technique for Simulating MOS Integrated Circuits. *IEEE Transactions on Circuits and Systems*, pages 1202–1205, June 1982.

[Luk75] Y.L. Luke. *Mathematical Functions and their Approximation.* Academic Press, New York, 1975.

[LV90] L.A. Lopez and K.A. Valimohamed. Software Environment for Implementing Engineering Applications on MIMD Computers. *Engineering with Computers*, 6:195–210, 1990.

[LW92] K-J. Lin and C-W. Wu. Realisation of Pipelined Mesh Algorithms on Hypercubes. *IEE Proceedings-E*, 139(3):189–194, May 1992.

[Mal82] W. Maly. An Approach to Yield Optimization by a Sample Neighbourhood Method. In *IEEE*, page 202, 1982.

[Mal89] W. Maly. Feasibility of Large Area Integrated Circuits. In *Wafer Scale Integration*, pages 31–56, Boston, 1989. Kluwer Academic Publishers.

[Mal90] W. Maly. Computer-Aided Design for VLSI Circuit Manufacturability. *Proceedings of the IEEE*, 78(2):356–392, February 1990.

[Mar82] G.I. Marchul. *Methods of Numerical Mathematics.* Springer-Verlag, New York, 1982.

[Maz92] P. Mazumder. Layout Optimisation for Yield Enhancement in on-chip-VLSI/WSI parallel processing. In *IEE Proceedings-E*, volume 139, pages 21–28, January 1992.

[MC80] Carver Mead and Lynn Conway. *Introduction to VLSI Systems.* Addison-Wesley, 1980.

[McG80] T.P. McGarty. *Stochastic Systems and State Estimation.* John Wiley, New York, 1980.

[McM82] S. McMinn. Semiconductor Manufacturing Considerations for VLSI Designers. *VLSI Design*, pages 16–18, July 1982.

[MD83] L. Mei and R.W. Dutton. Technology Modeling for VLSI Devices. *Solid State Technology*, pages 139–143, June 1983.

[MF86a] D. May and T. Fuge. The T800 Transputer. In *Electronics*, November 1986.

[MF86b] V. Mulitinovic and D. Fura. An Introduction to GaAs Microprocessor Architecture for VLSI. *Computer*, pages 31–41, March 1986.

[MF91] M. Maresca and T.J. Fountain. Massively Parallel Computers. volume 79, pages 395–401, April 1991.

[MG89] C. McCreary and H. Gill. Automatic Determination of Grain Size for Efficient Parallel Processing. *Communications of the ACM*, 32(9):1073–1078, September 1989.

[MH89] T. Merrow and N. Henson. System Design for Parallel Computing. *High Performance Systems*, pages 36–44, January 1989.

[Mir81] W.L. Miranker. *Numerical Methods for Stiff Equations.* D. Reidel, Holland, 1981.

[MM83] J.V. McCanny and J.G. McWhirter. Yield Enhancement of bit-level Systolic Array Chips using Fault Tolerant Techniques. *Electronics Letters*, 19(14):525–526, July 1983.

[MP80] O. Moeschlin and D. Pallascke, editors. *Game Theory and Mathematical Economics.* North-Holland, Amsterdam, 1980.

[MR80] E.F. Mishchenko and N.Kh. Rozov, editors. *Differential Equations with Small Parameters and Relaxation Oscillations.* Plenum Press, New York, 1980.

[MS90] P.K. Mozumder and A.J. Strojwas. Statistical Control of VLSI Fabrication Processes. *Proceedings of the IEEE*, 78(2):436–455, February 1990.

[MSB88] P.K. Mozumder, A.J. Strojwas, and D. Bell. Statistical Process Simulation for CAD/CAM. In *Proceedings of the IEEE 1988 Custom INtegrated Circuits Conference*, pages 13.5.1–13.5.4, May 1988.

[MSD86] W. Maly, A.J. Strojwas, and S.W. Director. VLSI Yield Prediction and Estimation: A Unified Framework. *IEEE Transactions on Computer-Aided Design*, CAD-5(1):114–130, January 1986.

[Mur82] S. Muroga. Fabrication and Cost Analysis. In *VLSI System Design - When and How to Design Very-Large-Scale Integrated Circuits.* John Wiley and Sons, 1982.

[Mur90] O.J. Murphy. Nearest Neighbour Pattern Classification Perceptrons. In *Proceedings of the IEEE*, volume 78, pages 1595–1597, 1990.

[Nev89] O. Nevanlinna. Remarks on Picard-Lindelof Iteration. *BIT*, 29:328–346, 1989.

[New79] A.R. Newton. Techniques for the Simulation of Large-Scale Integrated Circuits. *IEEE Transactions on Circuits and Systems*, CAS-26(9):741–749, September 1979.

[NPSV89] A.R. Newton, D.O. Pederson, and A. Sangiovanni-Vincentelli. *SPICE 3B1 User's Guide.* University of California, August 1989.

[NSD86] S.R. Nassif, A.J. Strojwas, and S.W. Director. A Methodology for Worst-Case Analysis of Integrated Circuits. *IEEE Transactions on Computer-Aided Design*, CAD-5:104–113, January 1986.

[NSV84] A.R. Newton and A.L. Sangiovanni-Vincentelli. Relaxation-based Electrical Simulation. *IEEE Transactions on Computer-Aided Design*, CAD-3(4):308–331, October 1984.

[OCD89] P. Odent, L. Claesen, and H. De Man. Feedback Loops and Large Subcircuits in the Multiprocessor Implementation of a Relaxation-Based Circuit Simulator. In *Proceedings of the 26th Design Automation Conference*, pages 25–30, June 1989.

[OHH89] D. Overhauser, I. Hajj, and Y. Hsu. Automatic Mixed-Mode Timing Simulation. In *Proceedings of the International Conference on Computer-Aided Design*, pages 84–87, November 1989.

[ON86] P. Odryna and S. Nassif. The ADEPT Timing Simulation Algorithm. *VLSI Systems Design*, pages 24–34, mar 1986.

[OPP91] L.F. Ortiz, R.Y. Pinter, and S.S. Pinter. An Array Language for Data Parallelism: Definition, Compilation and Applications. *Journal of Supercomputing*, 5(1):7–30, June 1991.

[Pap91] A. Papoulis. *Probability, Random Variables and Stochastic Processes.* McGraw-Hill, New York, 1991.

[Pel83] D.L. Peltzer. Wafer-Scale Integration: The Limits of VLSI? *VLSI Design*, pages 43–47, September 1983.

[PL83] D. Parkinson and H.M. Liddell. The Measurement of Performance on a Highly Parallel System. *IEEE Transactions on Computers*, C-32(1):32–40, January 1983.

[Poi83] C.J. Poirier. *QRS User's Guide.* Microelectronics Centre of North Carolina, 1983.

[PR82] D.J. Pradhan and S.M. Reddy. A Fault-Tolerant Communication Architecture for Distributed Systems. *IEEE Transactions on Computers*, C-31(9):863–870, September 1982.

[Pri90] J.F. Prins. A Framework for Efficient Execution of Array-Based Languages and SIMD Computers. *Third Symposium on the frontiers of Massively Parallel Computers*, page ??, 1990.

[PW89] R.S. Peipho and W.S. Wu. A Comparison of RISC Architectures. *IEEE Micro*, 9(4):51–62, August 1989.

[RD93] A.E. Ruehli and G.S. Ditlow. Circuit Analysis, Logic Simulation and Design Verification for VLSI. *Proceedings of the IEEE*, 71(1):34–143, January 1093.

[Ren89] P.V. Renterghem. Transputers for Industrial Applications. *Concurrency: Practice and Experience*, 1(2):135–147, December 1989.

[RM83] J.E. Requa and J.R. McGraw. The Piecewise Data Flow Architecture: Architectural Concepts. *IEEE Transactions on Computers*, C-32(5):425–437, May 1983.

[Rob87] F. Robert. *Discrete Iterations: A Metric Study.* Springer-Verlag, New York, 1987.

[RS90] S. Ranka and S. Sahni. Computing Hough Transforms on Hypercube Multicomputers. *Journal of Supercomputing*, 4(2):169–190, June 1990.

[RSV86] D. Riley and A. Sangiovanni-Vincentelli. Models for a New Profit-Based Methodology for Statistical Design of Integrated Circuits. *IEEE Transactions on Computer-Aided Design*, CAD-5(1):131–151, January 1986.

[Rut89] R.A. Rutenbar. Simulated Annealing Algorithms: An Overview. *IEEE Circuits and Devices Magazine*, pages 19–26, January 1989.

[SAFT92] G.B. Steven, R.G. Adams, P.A. Findlay, and S.A. Trainis. iHARP: A Multiple Instruction Issue Processor. In *IEE Proceedings-E*, volume 139, pages 439–452, September 1992.

[SB88] L. Soule and T. Blank. Parallel Logic Simulation on General Purpose Machines. In *Proceedings of the 25th Design Automation Conference*, pages 166–171, June 1988.

[Sca91] D.H. Scaefer. The Characterization and Representation of Massively Parallel Computing Structures. *Proceedings of the IEEE*, 79(4):461–479, April 1991.

[SCH86] J.P. Spoto, W.T. Coston, and C.P. Hernandez. Statistical Integrated Circuit Design and Characterization. *IEEE Transactions on Computer-Aided Design*, CAD-5(1):90–103, January 1986.

[Sch89] H. Schroder. Top-Down Design of Instruction Systolic Arrays fpr Polynomial Interpolation and Evaluation. *Journal of Parallel and Distributed Computing*, 6:692–703, 1989.

[Sch91] K.R. Schneider. A Remark on the Wave-Form Relaxation Method. *International Journal of Circuit Theory and Applications*, 19:101–104, 1991.

[SD86] C.J.B. Spanos and S.W. Director. Parameter Extractionfor Statistical IC Process Characterization. *IEEE Transactions on Computer-Aided Design*, CAD-5(1):66–78, January 1986.

[SF53] F.S. Shaw and S. Frederick. *An Intriduction to Relaxation Methods*. Dover Publications, New York, 1953.

[SG82] E.E. Swartzlander and B.K. Gilbert. Supersystems: Technology and Architecture. *IEEE Transactions on Computers*, C-31(5):399–409, May 1982.

[SGC+89] R.A. Saleh, K.A. Gallivan, M-C. Change, I.N. Hajj, D. Smart, and T.N. Trick. Parallel Circuit Simulation on Supercomputers. *Proceedings of the IEEE*, 77(12):1915–1931, December 1989.

[She91] W. Shen. Systolic Arrays for Multidimensional Discrete Transforms. *Journal of Supercomputing*, 4(1):1–16, January 1991.

[Shu83] M. Shubik. *Game Theory in the Social Sciences: Concepts and Solutions.* MIT Press, 1983.

[SI91] D. Smitley and K. Iobst. Bit-Serial SIMD on the CM-2 and the CRAY-2. *Journal of Parallel and Distributed Computing*, 11:135–145, 1991.

[Ski88] D.B. Skillicorn. A Taxonomy for Computer Architectures. *Computer*, 21(11):46–57, November 1988.

[Ski91] D.B. Skillicorn. Models for Practical Parallel Computation. *International Journal of Parallel Programming*, 20(2):133–158, 1991.

[SM71] A.P. Sage and J.L. Melsa. *Estimation Theory: with Applications to Communications and Control.* McGraw-Hill, New York, 1971.

[SO86] M.A. Styblinski and L.J. Opalski. Algorithms and Software Tools for IC Yield Optimization Based on Fundamental Fabrication Parameters. *IEEE TRansactions on Computer-Aided Design*, CAD-5(1):79–89, January 1986.

[Sou43] R.V. Southwell. On Relaxation Methods: a Mathematics for Engineering Science. *Proceedings of the Royal Society of London: Series A*, 184:253–288, 1943.

[SR83] M.A. Styblinski and A. Ruszczynski. Stochastic Approximation Approach to Statistical Circuit Design. *Electronics Letters*, 19(8):300–301, April 1983.

[SR90] J.J. Shynk and S. Roy. Convergence Properties and Stationary Points of a Perceptron Learning Algorithm. In *Proceedings of the IEEE*, volume 78, pages 1599–1603, 1990.

[SSV86] A.J. Strojwas and A. Sangiovanni-Vincentelli. Foreward to Statistical Techniques in VLSI Design Issue. *IEEE Transactions on Computer-Aided Design*, CAD-5(1):1–2, January 1986.

[Ste86] M.L. Stein. An Efficient Method of Sampling for Statistical Circuit Design. *IEEE Transactions on Computer-Aided Design*, CAD-5(1):23–29, January 1986.

[Sti91] L. Stiller. Group Graphs and Computational Symmetry on Massively Parallel Architecture. *Journal of Supercomputing*, 5(2):99–118, October 1991.

[Str89] A.J. Strojwas. Design for Manufacturability and Yield. In *Proceedings of the 26th Automation Conference*, pages 454–459, 1989.

[SW92] V.L. Smith and A.W. Williams. Experimental Market Economics. *Scientific American*, 77(12):72, December 1992.

[SY90] A.D. Singh and Y. Youn. Defect Tolerance Scheme for gigaFlop WSI Architectures. *Proceedings of the International Conference on Wafer Scale Integration*, pages 109–115, 1990.

[Sze83] Sze. *VLSI Technology.* McGraw Hill, 1983.

[Tak62] L. Takacs. *Introduction to the Theory of Queues.* Oxford University Press, New York, 1962.

[Tam91] K. Tamura. High Speed Computing System for Scientific and Technological Uses. *Journal of SuperComputing*, 5:157–168, 1991.

[TC86a] D. Tsao and C. Chen. A Fast-Timing Simulator for Digital MOS Circuits. *IEEE Transactions on Computer-Aided Design*, CAD-5(4):536–540, October 1986.

[TC86b] B-Y. Tsaur and C.K. Chen. Submicrometer CMOS Devices in Zone-Melting-Recrystallized SOI Films. *IEEE Electron Device Letters*, EDL-7(7):43–445, July 1986.

[Tej85] E.R. Teja. Distributed Supermicro Architectures tackle Computationally Intensive Chores. *EDN*, pages 51–54, May 1985.

[TMVK92] E.H. Tizwell, S.A. Matar, D.A. Voss, and A.Q.M. Khaliq. Explicit Numerical Methods with Enhanced Stability Properties for First-Order Autonomous Initial-Valued Problems. *International Journal of Engineering Science*, 30(3):379–392, 1992.

[Vau92] J.G. Vaughan. Static Performance of a Divide and Conquer Information-Distribution Protocol Supporting a Load-Balancing Scheme. In *IEE Proceedings-E*, volume 139, pages 430–438, September 1992.

[VD90] H.A. Vander. Vorst and P.Van. Dooren. Parallel algorithms for numerical linear algebra. In *Advances in Parallel Computing*, Amsterdam, 1990. North-Holland.

[VP82] A. Vladimirescu and D.O. Pederson. Performance Limits of the Classie Circuit Simulator Program. *IEEE Transactions of Circuits and Systems*, pages 1229–1232, 1982.

[Wer84] J. Werner. A System engineer's Guide to Simulators. *VLSI Design*, pages 27–31, February 1984.

[WG83] R.M. Warner and B.L. Grung. *Transistors: Fundamentals for the Integrated Circuit Engineer.* John Wiley, New York, 1983.

[Won71] E. Wong. *Stochastic Processes in Information and Dynamical Systems.* Springer-Verlag, New York, 1971.

[WTHP91] N.B. Wilding, A.S. Trew, K.A. Hawick, and G.S. Pawley. Scientific Modeling with Massively Parallel SIMD Computers. *Proceedings of the IEEE*, 79(4):574–585, April 1991.

[WYC87] K.C.K. Weng, P. Yang, and J.H. Chern. A Predictor/CAD Model for Buried-Channel MOS Transistors. *IEEE Transactions on Computer-Aided Design*, CAD-6(1):4–16, January 1987.

[YC82] P. Yang and P.K. Chatterjee. SPICE Modelling for Small Geometry MOSFET Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, CAD-1(4):169–182, October 1982.

[YHC+86] P. Yang, D.E. Hocevar, P.F. Cox, C. Machala, and P.K. Chatterjee. An Integrated and Efficient Approach for MOS VLSI Statistical Design. *IEEE Transactions on Computer-Aided Design*, CAD-5(1):5–14, January 1986.

[Zia92] S.G. Ziavras. On the Problem of Expanding Hypercube-Based Systems. *Journal of Parallel and Distributed Computing*, 16:41–53, 1992.