Parallel Computation of Non-Deterministic Algorithms in VLSI

by

Peter D. Hortensius

A thesis
presented to the University of Manitoba
in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy
in
Electrical Engineering

Winnipeg, Manitoba, 1987

ISBN 0-315-44222-0

PARALLEL COMPUTATION OF NON-DETERMINISTIC ALGORITHMS IN VLSI

BY

PETER D. HORTENSIUS

A thesis submitted to the Faculty of Graduate Studies of the University of Manitoba in partial fulfillment of the requirements of the degree of

DOCTOR OF PHILOSOPHY

I hereby declare that I am the sole author of this thesis.

I authorise the University of Manitoba to lend this thesis to other institutions or individuals for the purpose of scholarly research.

Peter D. Hortensius

I further authorise the University of Manitoba to reproduce this thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

Peter D. Hortensius

The University of Manitoba requires the signatures of all persons using or photocopying this thesis. Please sign below and give address and date.

Name	Address	Date

Abstract

This work examines parallel VLSI implementations of nondeterministic algorithms. It is demonstrated that conventional pseudorandom number generators are unsuitable for highly parallel applications. For example, while linear feedback shift registers (LFSR) are adequate for generation of single pseudorandom bit streams, the bit streams from different cells in the LFSR are highly correlated. Efficient parallel pseudorandom sequence generation can be accomplished using certain classes of elementary one-dimensional cellular automata (two binary states per site and only nearest neighbour connections). The pseudorandom numbers appear in parallel from various cells in the cellular automaton on each clock cycle. Extensive study of the properties of these new pseudorandom number generators is made using standard empirical random number tests, cycle length tests, and implementation considerations.

Furthermore, it is shown these particular one-dimensional cellular automata can form the basis of efficient VLSI architectures for computations involved in the Monte Carlo simulation of both the percolation and Ising models from statistical mechanics. The architectures provide a spatially-distributed set of pseudorandom numbers which are required in the local nondeterministic decisions at the various sites in the array. It is shown that the time-intensive task of sampling the percolation and Ising configurations is expedited by the inherent parallelism of this approach. The architectures can be used to report pertinent information such as the magnetisation to a host computer. It is demonstrated that these architectures can provide speedup of several orders of magnitude over conventional Monte Carlo simulation. For example, a 1000×1000 Ising lattice can be completely updated in less than 1 μsec. The correctness of this approach is verified by computer simulation of the new architectures which derived the correct critical exponents for both the percolation and Ising models.

Finally, a variation on a Built-In Self-Test technique is presented. It is based upon a distributed pseudorandom number generator derived from a one-dimensional cellular automata array. These Cellular Automata-Logic-Block-Observation (CALBO) circuits improve upon conventional design for testability circuitry such as BILBO as a direct consequence of reduced cross-correlation between the bit streams which are used as inputs to the logic unit under test. This approach answers the problems arising from the correlated bit streams produced by the conventional LFSR. In addition, it is noted these cellular automata implementations exhibit locality and topological regularity; important attributes for a VLSI implementation. It is shown that much of the analysis of pseudorandom testing is more directly applicable to CALBO-based pseudorandom testing as compared to LFSR-based schemes, in that statistical assumptions regarding the pseudorandom test vector generation are better justified in the former case. The data compaction properties of CALBO are examined and it is found that cyclic group rule cellular automata provide comparable signature analysis properties to the LFSR. An important feature still to be fully investigated is the possibility that some cellular automata may be able to generate weighted pseudorandom test patterns.

Acknowledgements

The author wishes to thank Dr. Howard C. Card for his patient supervision and encouragement throughout the time of this work. In addition, Dr. Bob McLeod's availability and assistance are also gratefully recognised.

The author also wishes to express his appreciation to his colleagues in the VLSI Research Laboratory at the University of Manitoba, especially Chris and Roland Schneider and Werner Pries.

Financial assistance from the Natural Sciences and Engineering Research Council Postgraduate Scholarship program and the Province of Manitoba Strategic Grants program is acknowledged.

Equipment support provided by the Natural Sciences and Engineering Research Council through the Canadian Microelectronics Corporation and integrated circuit fabrication facilities provided by Northern Telecom Ltd. are recognised.

Finally, the assitance of CANCAD Technology Corp. is also recognised for the use of their three-dimensional plotting package.

To Gloria

Table of Contents

List of Figures

List of Tables

Chapter 1
Introduction

1.1. MOTIVATION

This thesis is motivated by the potential for massively parallel VLSI systems to study large problems. These problems are often best solved by employing algorithms with nondeterministic components. In this study we restrict ourselves to nondeterministic algorithms which can be implemented on a parallel system using a single instruction multiple data, or SIMD, architecture. Many problems, especially important modelling problems from statistical mechanics, fall into this class.

Solution of large problems using nondeterministic algorithms often involves tradeoffs between interesting problem sizes and computationally reasonable solution times. However, even for small problem sizes many of these nondeterministic problems stretch computer resources to the limit. The salient features of a nondeterministic algorithm include generating a random number, comparing it to some probability, and taking appropriate action, usually some simple operation. Therefore, on most computer systems we are restricted by the rate at which we can generate pseudorandom numbers. The most obvious solution to increase computational throughput, other than technological improvement of the computing hardware, is to attempt to implement portions of the algorithm in parallel. However, this creates the need for efficient parallel pseudorandom number generation. In a VLSI implementation all circuits must be both area and time efficient or precious silicon area and/or computation time will be wasted. Thus, for parallel VLSI implementations of nondeterministic algorithms we require an *area-time* efficient pseudorandom number generator. Therefore, while in this work we primarily consider the VLSI implementations of nondeterministic algorithms, we must first concern ourselves with the development of a suitable pseudorandom number generator for such a parallel computing environment.

A second motivation is to examine the suitability of current pseudorandom test pattern generators and signature analyzers for built-in self-test (BIST) of VLSI circuits. In this problem we are again concerned with the generation of pseudorandom numbers. However, we are not concerned with the generation of these pseudorandom numbers in parallel but rather with generating pseudorandom numbers at high speed using minimal area. It is known that the most common mechanism for generating pseudorandom test patterns and performing signature analysis for BIST (the linear

feedback shift register) suffers from cross-correlation. This is especially problematic in a BIST environment since some circuit faults cannot be detected using such test pattern generators and signature analyzers. Therefore, we seek to propose improved pseudorandom test pattern generators and signature analyzers for BIST.

1.2. OBJECTIVES

The objective of this thesis is twofold. First, we seek to study the VLSI implementation of massively parallel computing systems for the solution of nondeterministic algorithms. Secondly, we attempt to derive improved BIST circuits for both test pattern generation and signature analysis. Consequently, we must place a large emphasis on the requirements of pseudorandom number generation for such systems. In this light we seek to discover an *area-time* efficient pseudorandom number generator. The resulting pseudorandom number generator's suitability is demonstrated by first studying two highly parallel nondeterministic algorithms. These algorithms arise in statistical mechanics where massively parallel Monte Carlo simulation of the percolation and Ising models could result in a speedup by several orders of magnitude over conventional serial Monte Carlo simulation. Secondly, we examine the application of the new pseudorandom number generator to random testing of digital VLSI circuits. In this work we must show that the new test pattern generator and signature analyzer is both small and fast when compared to conventional techniques. We must also derive measures by which performance comparisons to other techniques can be made.

1.3. PRINCIPAL CONTRIBUTIONS OF THIS THESIS

This thesis concerns itself with the study of *area-time* efficient hardware pseudorandom number generation, parallel computation of nondeterministic algorithms, pseudorandom test pattern generation and signature analysis for digital circuit testing. Contributions arising from this work include:

1. An analysis of conventional pseudorandom number generation with respect to its implementation in a fine grained parallel processing environment is presented.
2. A new area-time efficient pseudorandom number generator based on simple one-dimensional cellular automata is demonstrated.
3. A parallel architecture for simulation of the lattice percolation model of statistical mechanics is described. Correctness of the design was verified by simulations of the architecture which produced the correct critical exponents associated with phase transitions in the model system. Speedup over conventional simulation techniques is shown to be $O(N)$, where N is the number of sites in a lattice.
4. Two architectures for high speed simulation of the Ising model are presented. As with the percolation model computer, correctness of the design was verified by simulation.

5. A novel cellular automata-based pseudorandom test pattern generator is described which overcomes some of the fault coverage problems in the traditional linear feedback shift register-based pseudorandom test pattern generator. The topological regularity of the new generator facilitates easy register width changes and so therefore is well suited to incorporation in computer-aided design software tools, as well as being an efficient basis for self-testing hardware.
6. A first study of the signature analysis properties of simple one-dimensional cellular automata has been made. Results indicate that cellular automata whose rules display cyclic group properties are well suited for use in signature analysis with comparable aliasing properties to the linear feedback shift register.

1.4. THESIS ORGANISATION

The thesis is organised into four sections. Chapter 2 deals with the subject of random number generation with emphasis on applications in parallel processing with fine-grained processors. A working definition of *randomness* based on standard random number tests is derived and several conventional pseudorandom number generators are tested and examined in the light of parallel processing applications. Finally, a novel *area-efficient* pseudorandom number generator based upon cellular automata is demonstrated and tested. Chapters 3 and 4 explore two potential applications for this new generator. In Chapter 3 we are concerned with two nondeterministic algorithms used in statistical mechanics. Parallel architectures are derived which show great promise for use as special hardware accelerators for the percolation and Ising models. The correctness of the approach is shown by simulation of the architectures with respect to generating the correct model parameters, or critical exponents. Chapter 4 considers a different nondeterministic problem, that of built-in testing of VLSI circuits. Here new built-in self-test circuits based on the cellular automata-based pseudorandom number generators of Chapter 2 are used as a logic block observers. Measures of the fault detection and signature analysis properties are derived and compared with linear feedback shift register based circuits. Finally, Chapter 5 presents some conclusions and suggestions for further work.

There are three appendices to this work. Appendix A derives a confidence estimate of the probability of generating a *nonrandom* sequence, given that a pseudorandom number generator has passed the random number tests. Complete cycle length properties of the class 3 cellular automata studied in Chapter 2 are given in Appendix B. Appendix C presents detailed information on the weight of each output bit for all possible simple one-dimensional cellular automata in the context of weighted pattern generation for built-in self-test.

An accompanying volume presents detailed information on the computing system and programs used in this work.

Chapter 2
Parallel Pseudorandom Number Generation in VLSI

2.1. INTRODUCTION

In this chapter the efficient generation of random numbers by deterministic methods for use by parallel processors in fine-grained VLSI arrays, such as the two-dimensional mesh architecture shown in Fig. 2.1, is discussed. By fine-grained it is implied that the individual processors do not have a great deal of processing power, but instead, the architecture relies on the large number of these processors to create a powerful machine. In the execution of algorithms having nondeterministic components, this means that most processors will consist of a software or hardware based random number generator (RNG) and some simple processing elements. For example, in a parallel sampling algorithm, it is possible that the processor will consist solely of a data selector, possibly from a small memory, and a comparator to compare the selected value with a random number. In these architectures, which have been used to solve problems based on the percolation and Ising models of statistical mechanics (see Chapter 3), it is necessary for the random numbers at each processor to be available on each clock cycle to achieve maximum throughput at each processing site. This requires that special hardware be dedicated to random number generation at each processor. To provide parallel hardware random number generation two methods are possible. Firstly, one may employ a technique whereby a large global random number is generated on each clock cycle and a local random number for each processing site is obtained by selecting only a small number of bits from this global random number. This method may be employed only if the bits in each local number are uncorrelated and if there is also no cross-correlation between local random numbers at neighbouring processing sites. Secondly, one could have a RNG at each processing site.

In this chapter the requirements of parallel computer architectures which require random number generation will be first examined using conventional RNGs. Special emphasis will be placed on the VLSI implementation of these architectures including estimates of processor area, A, and computation time, T.[2.1] It will be shown that conventional RNGs are inefficient in terms of area and time requirements for the fine-grained parallel processing environment of Fig. 2.1. Novel architectures will then be proposed, based on a new RNG which uses cellular automata. These architectures solve many of the problems associated with using conventional techniques of RNG in

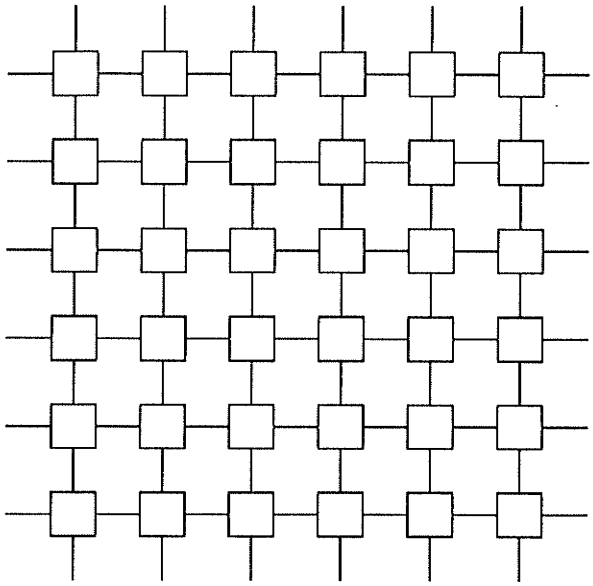


Figure 2.1 : *A fine-grained parallel processing two-dimensional mesh architecture.*

fine-grained parallel processing machines.

These architectures have applications in the efficient implementation of parallel nondeterministic algorithms such as modelling schemes based upon statistical mechanics, hardware accelerators for simulated annealing, and other parallel Monte Carlo simulations. Another important application of the results presented in this chapter lies in the area of built-in self-test circuits (BIST) for VLSI where an improved pseudorandom number generator (PRNG) as compared to the traditional linear feedback shift register (LFSR) is demonstrated. These applications are explored in Chapters 3 (Parallel Architectures for Statistical Mechanics) and 4 (Applications to Built-In Self-Test).

2.2. DEFINITION OF RANDOMNESS

It is very difficult to discuss the idea of randomness without being drawn into some sort of philosophical discussion about what *random* means. An interesting and thoughtful discussion on this topic can be found in [Knuth1981]. It is inappropriate to consider an individual number to be random; instead, a sequence of numbers must be considered. Here we are concerned with the generation of random number sequences

[2.1] This allows us to make comparisons on the relative performance of each architecture using the methods of [Thompson1980]. The area and time estimates will be based on a 3 micron single metal twin tub CMOS process technology so that it will be possible to insert the constants required to make exact comparisons and not just asymptotic ones. All implementations consist of static circuits and an 8 bit word size. Relative comparisons are expected to be reasonably technology independent.

by deterministic methods. The resulting sequences cannot be completely random since they are generated deterministically, but are rather, referred to as pseudorandom sequences. The definition of a pseudorandom sequence to be used here will consider a sequence to be pseudorandom if it satisfies several standard empirical random number tests as described below.[2.2] For these tests a general form for a pseudorandom sequence is assumed to be

$$<Z_i> = Z_0, Z_1, Z_2, \cdots \tag{2.1}$$

where each Z_i is a real number between zero and one. This work is concerned with the hardware generation of pseudorandom sequences which are most conveniently expressed in terms of integers; an integer valued sequence of the form

$$<X_i> = X_0, X_1, X_2, \cdots \tag{2.2}$$

where each

$$X_i = \lfloor d\ Z_i \rfloor \tag{2.3}$$

will be employed rather than the real-valued sequence of Eqn. 2.1. The sequence of integers will have values between 0 and $d-1$ giving an integer word size of $\lceil \log_2 d \rceil$ bits.

Many of the following tests perform some operation using the sequence, usually in the nature of a counting operation, and the results are then compared to a given distribution for a particular test. The question that naturally arises is how can it be determined if the results of the test are the same as what is desired, or expected, of a pseudorandom sequence? Obviously we do not expect the results to be exactly the same since it is highly unlikely that a truly random sequence would exhibit ideal properties. On the other hand, deviation too far from the ideal is also unsatisfactory. Therefore, the comparison must be made for a range of acceptable behaviour. The comparison technique used here is the well known chi-square test (χ^2 test). In this test each possible event is assigned a probability, p_s; then the number of occurrences of that event are counted, Y_s. After all the events have been counted each Y_s is compared to the expected number of occurrences, np_s, where n is the number of elements in the test. If we sum the squares of the difference between Y_s and np_s for each possible event we determine a measure of the difference between the expected and actual results. This quantity, usually labelled as V, can be expressed

[2.2] Tests 1 through 9i are primarily adapted from Knuth [Knuth1981] with some, minor changes and additions. Other references detailing similar test procedures for sequences of pseudorandom numbers are available in [Kendal1938], [Lewis1969], [MacLaren1965], [Marsaglia1968], and [Payne1971]. Test 9ii is newly proposed in this work, although the problem of independence between bits within a random number has been previously considered by others (see for example [Tootill1973]). Other tests specific to a given type of PRNG have been proposed [Marsaglia1984]. It is not expected that other tests will affect the results of this work since the tests of Knuth are generally considered to be adequate for most applications requiring pseudorandom sequences.

mathematically as

$$V = \frac{(Y_1 - np_1)^2}{np_1} + \frac{(Y_2 - np_2)^2}{np_2} + \cdots + \frac{(Y_E - np_E)^2}{np_E} \tag{2.4}$$

for a test with E events. Equation 2.4 can be more conveniently expressed as

$$V = \frac{1}{n} \sum_{s=1}^{E} \left\{ \frac{Y_s^2}{p_s} \right\} - n \ . \tag{2.5}$$

The quantity V is then compared to tables which show the probability of obtaining a value less than, or equal to, V for a given number of events, or degrees of freedom. Therefore, the χ^2 test returns a probabilistic result as to the randomness of the sequence; i.e. there is no yes-no answer.

Knuth [Knuth1981] suggests a method of determining if V is a reasonable value by assigning probabilities to various values of V corresponding to the probability that a random sequence would produce a value less than, or equal to, a given value of V. These results are usually presented in table form for convenience. If the value of V is greater than the 99% table entry or less than the 1% table entry then the χ^2 test has been failed. If the value of V falls in the 1% to 5% or 95% to 99% range then the value of V is suspect and in the range 5% to 10% or 90% to 95% V is considered possibly acceptable. Only in the range 10% to 90% is V definitely considered to be a value that might be produced by a pseudorandom sequence. The problem with Knuth's pass-fail method is that it is a soft measure since it has four different categories, or results (failed, almost failed, almost passed, and passed).

In this work computer based testing will be used to verify the randomness of generated sequences. Hence a method which provides strict pass and fail ranges is more desirable. One such method which quickly determines if V is an acceptable value is that V must lie in the range $d \pm 2\sqrt{d}$ [Sedgewick1983]. This gives an approximate pass range of 7.5% to 92.5% with the fail range consisting of 0.0% to 7.5% and 92.5% to 100%. This method will be used here because of its ease of use in computer based random number testing.[2.3] It should be noted that the χ^2 test is valid if, and only if, the data, in this case event counts in the test categories, are independent and the value of n is large. The larger the value of n the more accurate the test, however, if n is too large then locally nonrandom behaviour will be indistinguishable since it will be washed out by the global properties of the sequence. A general rule of thumb for n is that $n \geq 10E$ [Sedgewick1983].

[2.3] That a particular sequence yields an unsatisfactorily low or high value of V does not automatically indicate the generator as being nonpseudorandom but rather that a particular sequence from that generator failed. In fact, it is expected that some random sequences will, on occasion, fail the χ^2 test. Therefore, only if sequences consistently fail the χ^2 test, can the generator of those sequences be said to be producing nonpseudorandom sequences.

2.2.1. Random Number Tests

1) **Equidistribution test (Frequency test).** The sequence must consist of numbers which are uniformly distributed between zero and $d-1$ (i.e. $p_s = 1/d$ and $E = d$).

2) **Serial test.** Successive pairs of numbers in the sequence should be uniformly distributed and independent, i.e. successive pairs (X_{2j}, X_{2j+1}) should be equidistributed between (0, 0) and $(d-1, d-1)$ (here $p_s = 1/d^2$ and $E = d^2$). This test can be extended to triples $(X_{3j}, X_{3j+1}, X_{3j+2})$ ($p_s = 1/d^3$ and $E = d^3$) and quadruples $(X_{4j}, X_{4j+1}, X_{4j+2}, X_{4j+3})$ ($p_s = 1/d^4$ and $E = d^4$). However, tests using sets greater than quadruples have such a large number of categories ($\geq d^5$) that it may not be possible, or convenient, to properly test the equidistribution of the categories. The value of d, or the modulus, also affects the extent of the test. For example, even a small modulus, d, will generate a large number of categories; e.g. $d = 5$ yields 625 categories for the quadruple serial test. This means that n should be ≥ 6250 for a good χ^2 test. For larger moduli very long sequences must be tested because of the correspondingly large number of categories. Thus, serial tests, especially those of higher order, are best used with small moduli.

3) **Gap test.** The length of the gaps between occurrences of numbers in the sequence that lie within a certain range is tabulated and the collection of lengths should then lie within a binomial distribution. Consider two values α and β, the lengths of consecutive numbers $X_j, X_{j+1}, \cdots, X_{j+r}$ in which $\alpha \leq X_{j+r} \leq \beta$, but each $X_{j+i} < \alpha$ or $X_{j+i} > \beta$, $i < r$, are considered. If we use gap lengths 0, 1, $\cdots$, t-1, and $\geq t$ we have probabilities $p_0 = p$, $p_1 = p(1-p)$, $p_2 = p(1-p)^2$, $\cdots$, $p_t = p(1-p)^t$, where $p = \dfrac{\beta - \alpha}{d}$ and $E = t + 1$.

4) **Poker test (Partition test).** The number of k-tuples of r different values over groups of k successive numbers is counted; these should follow a predetermined distribution [Knuth1981]. Here $E = d/k$ and

$$p_r = \frac{d(d-1)(d-2)\cdots(d-r+1)}{d^k} \left\{ {k \atop r} \right\} \tag{2.6}$$

where $\left\{ {k \atop r} \right\}$ is a Stirling number of the second kind.

5) **Coupon collector's test.** The length of sequence required to obtain a complete collection of numbers in the range 0 to $d-1$ is considered; this set of numbers must have a distribution of a particular form [Knuth1981]. Here we apply the χ^2 test to the set $\{C_d, C_{d+1}, \cdots, C_{d+t}\}$, where C_r, $d \leq r \leq t$, is the number of times r successive numbers were required to obtain a complete collection. The χ^2 test uses $E = t$ and

$$p_r = \frac{d!}{d^r} \left\{ {r-1 \atop d-1} \right\}, \; d \leq r < t \qquad p_t = 1 - \frac{d!}{d^{t-1}} \left\{ {t-1 \atop d} \right\}. \tag{2.7}$$

As in the serial test the value of d directly affects the length of sequence required for a good test. For example, if $d = 100$ and $t = 20$ then we must have a sequence length long enough to perform at least 200 tests (i.e. 20,000 numbers) for a good χ^2 test. Therefore, the coupon collector's test is best suited to small moduli.

6) **Permutation test.** The sequence is divided into successive blocks of length q. Each of the $q!$ possible orderings should occur with equal frequency (i.e. $p_s = 1/q!$ and $E = q!$). It should be noted that small moduli will invalidate higher order permutation tests (i.e. larger values of q). If the modulus for a sequence is small, there will be a tendency to have two or more equal values in a block thereby enhancing the probability of certain orderings over others. Experimentally it has been observed that the probability of two or more values in a block being equal should be less than 0.2 for a permutation test on size q blocks to behave as expected.

7) **Run tests.**

I) Run up: The lengths of blocks in the sequence which are monotonically increasing are counted; these must correspond to a given distribution [Knuth1981]. Each run is separated by one number (i.e. if $X_j > X_{j+1}$ then start next run with X_{j+2}) for independence, here $E = t + 1$ for runs of length 1, 2, $\cdots$, t, and $> t$ and

$$p_r = \frac{1}{r!} - \frac{1}{(r+1)!}, \; 1 \le r \le t \qquad p_{t+1} = 1 - \sum_{r=1}^{t} p_r \quad . \tag{2.8}$$

As for the permutation test, if small moduli are used, there is a tendency to have equal values within a block which will shorten the length of monotonically increasing blocks. This will increase the probability of short runs causing unexpected test results. Experimentally it has been observed that the run test requires the same probability of having two or more values equal in a given block as the permutation test to behave as expected (i.e. the probability of two or more values being equal in a block of length t should be less than 0.2).

II) Run down: The run up test is repeated for monotonically decreasing block lengths.

8) **Maximum of t test.** The sequence is broken up into successive blocks of length t; the maximum value of each block must lie in a power distribution (i.e. let $V_i = max(X_{ti}, X_{ti+1}, \ldots, X_{ti+t-1})$, $i = 0, 1, \cdots, \frac{n}{t} - 1$ then the sequence of values $<V_i>$ must be distributed with

$$p_s = \left(\frac{s+1}{d}\right)^t - \left(\frac{s}{d}\right)^t \tag{2.9}$$

and $E = d$.

9) **Correlation tests.** Two correlation tests are used to test for auto and cross-correlation in the sequence numbers:

I) Serial correlation: In this test the following autocorrelation statistic is calculated for autocorrelation amongst the numbers in the sequence

$$C_A = \frac{n(X_0X_1+X_1X_2+\cdots+X_{n-2}X_{n-1}+X_{n-1}X_0) - (X_0+X_1+\cdots+X_{n-1})^2}{n(X_0^2+X_1^2+\cdots+X_{n-1}^2) - (X_0+X_1+\cdots+X_{n-1})^2} \quad . \tag{2.10}$$

If we have two sequences $<X_i>$ and $<Y_i>$ both of length n we can compute a cross-correlation statistic between the numbers in the two sequences as

$$C_C = \frac{n\sum_{i=0}^{n-1}\left[X_iY_i\right] - \left[\sum_{i=0}^{n-1}X_i \cdot \sum_{i=0}^{n-1}Y_i\right]}{\sqrt{\left[n\sum_{i=0}^{n-1}X_i^2 - (\sum_{i=0}^{n-1}X_i)^2\right] \cdot \left[n\sum_{i=0}^{n-1}Y_i^2 - (\sum_{i=0}^{n-1}Y_i)^2\right]}} \tag{2.11}$$

Both the C_A and C_C correlation statistics lie in the range -1 to $+1$, with small values indicating independence between values in the sequence. Therefore, it is desirable to have both C_A and C_C close to zero. A *good* value for C_A will lie in the range $\mu_n \pm 2\sigma_n$ [Knuth1981], where

$$\mu_n = \frac{-1}{n-1}, \qquad \sigma_n = \frac{1}{n-1}\sqrt{\frac{n(n-3)}{n+1}}, \qquad n > 2 \quad . \tag{2.12}$$

II) Bit sequence correlation: Each number in the sequence $<X_i>$ can be represented using m bit binary notation (assuming an m bit word size). If we consider bit i in each sequence number's binary notation we can form a binary sequence, $<x_i>$. Correlation tests are then performed on the resulting m binary sequences. Here we compute the autocorrelation of a binary sequence to be

$$BC_A^t = \frac{\sum_{k=0}^{n-1}\left\{\left[x_i(k) - \mu_i\right]\left[x_i(k+t) - \mu_i\right]\right\}}{\sigma_i^2} \tag{2.13}$$

where

$$\mu_i = \frac{1}{n}\sum_{k=0}^{n-1}x_i(k) \qquad \sigma_i^2 = \frac{1}{n}\sum_{k=0}^{n-1}(x_i(k) - \mu_i)^2 \tag{2.14}$$

BC_A^t = autocorrelation of binary sequence $<x_i>$ with time displacement t;

$x_i(k)$ = k'th bit in the sequence $<x_i>$;

μ_i = mean of sequence $<x_i>$;
σ_i^2 = variance of sequence $<x_i>$;

and the cross-correlation of two binary sequences $<x_i>$ and $<x_{i+j}>$ as

$$BC_{C_j}^t = \frac{\sum_{k=0}^{n-1} \left\{ \left[x_i(k) - \mu_i \right] \left[x_{i+j}(k+t) - \mu_{i+j} \right] \right\}}{\sigma_i^2} \tag{2.15}$$

where

$BC_{C_j}^t$ = cross-correlation of binary sequences $<x_i>$ and $<x_{i+j}>$ with time displacement t.

As for the serial correlation test the values of BC_A^t and $BC_{C_j}^t$ lie in the range –1 to +1 and should be close to zero for independence. Note that the correlation of $<x_i>$ with itself (i.e. BC_A^0) must equal +1. For this work only the magnitude of the binary sequence correlation values will be considered. Hence the actual range of values will be from 0 to +1.

10) **Visual test.** The human visual process has an uncanny ability to detect patterns governed by long range correlations. In this test a graphical representation of the numbers in the sequence is plotted on a computer screen and emergent patterns, if any, are observed. There are two graphical representations used in this work. Firstly, each number in the sequence is considered as a binary word with each number corresponding to a unique horizontal line on the screen. Each line is then considered to consist of the same number of *pixel* divisions as the word size of the number. These divisions are then *turned on* or *off* depending on the value of the corresponding bit in the pseudorandom number. An example of this representation is given in Fig. 2.15.

The second form of representation is a raster scan of a binary sequence. Here each horizontal pixel corresponds to one bit in the binary sequence and is correspondingly turned on or off. An example is given in Fig. 2.2 where two binary sequences are plotted using the raster scan representation. Note that Fig. 2.2(a) has a discernible pattern and therefore must be considered a nonpseudorandom sequence while Fig. 2.2(b) does not appear to exhibit any pattern. The other tests, 1 to 9 described above, confirm that the sequence of Fig. 2.2(a) is not a pseudorandom sequence. Surprisingly, the sequence of Fig. 2.2(b) also turns out not to be a pseudorandom sequence, since it does not adequately pass all of the above tests. This shows that visual inspection may serve as a preprocessing filter in eliminating many nonpseudorandom sequences. However, the sequences which are visually acceptable must still be subjected to further tests.

The opposite situation can also arise in which sequences pass all the other random number tests but fail the visual test. The eye is able to detect patterns which are invisible to the tests 1 to 9 above. This case arises later in the chapter.

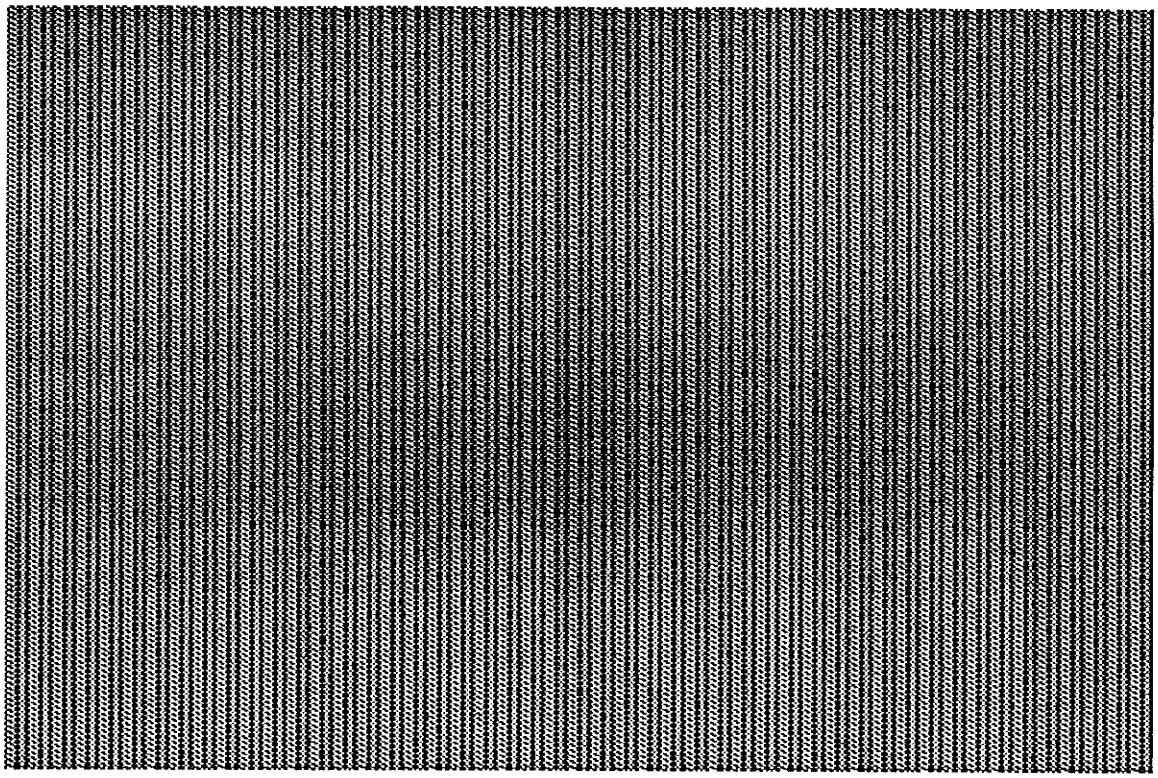


Figure 2.2 (a): *A visually unacceptable (i.e. nonpseudorandom) sequence.*

Note that this test differs from the unreliable test of examining the numbers for patterns [Knuth1981] since for the visual patterns to be detected, some observable quantity must be repeating over a large block of numbers in the sequence. However, in examining the numbers one can only observe very local phenomena and not the more important global behaviour.

The tests 1 to 9 above must be applied to the pseudorandom sequence in such a way that local and global randomness can be tested. For example, in a sequence of 100,000 numbers it is possible that a small collection of numbers could be decidedly nonrandom. If one examines all 100,000 numbers as a block then these few numbers could be overlooked since the sequence is much larger than the small anomaly. This is overcome by testing the sequence in large and small blocks to check for local properties. The 100,000 number sequence would first be tested as a whole to check for global properties, then it would be decomposed into 10 sequences of 10,000 numbers and each of these sequences would be tested. Each of the 10,000 number sequences would then be reduced to 10 sequences of 1000 numbers and each of these 1000 number sequences would be tested. This continues as smaller and smaller block lengths are used. The terminating block size used for the results presented here is 1000 since many of the tests fail the χ^2 test requirements of $n \geq 10E$ for lesser values of n. Another point to note is that some pseudorandom number tests are not valid for certain moduli (i.e. the value of d). Therefore, it is best to use at least two

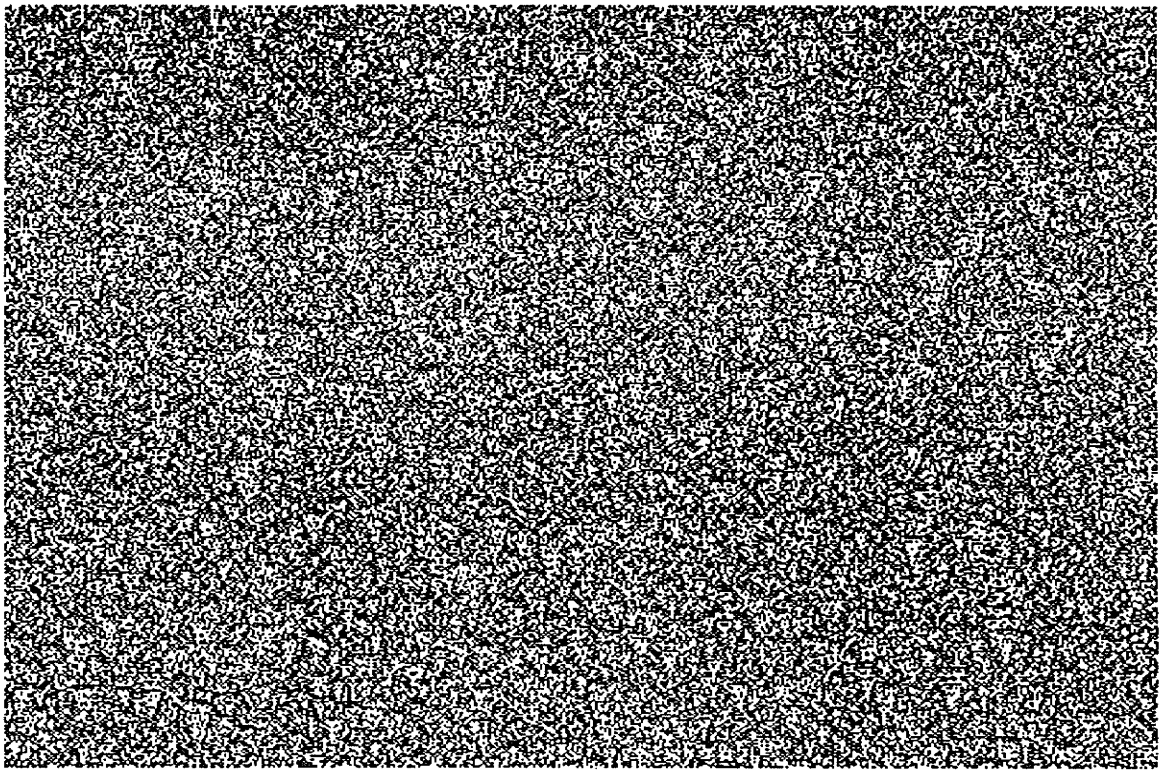


Figure 2.2 (b): *A visually acceptable pseudorandom sequence.*

different moduli, one large and the other small, in order to properly utilise the different random number tests.

Other powerful random number tests such as the *spectral test* [Coveyou1967], which considers the joint distribution of consecutive elements in a full-period pseudorandom sequence by measuring the distance between the most widely separated set of adjacent parallel hyperplanes in n-space; and the *lattice test* [Marsaglia1972], which considers the filling of an n-dimensional lattice by finding the most nearly orthogonal vectors for the lattice structure, the ratio of the longest basis vector to the shortest is used as the measure of acceptability. These tests have been used mainly with linear congruential generators, since they are amenable to the analysis required to formulate describing equations. The analysis of other generators, such as the cellular automata PRNGs described later in this chapter, by the spectral or lattice tests is left as an open problem.

Although the above tests will certainly analyse a sequence to detect deviations from pseudorandomness it would be preferable to develop a general theoretical basis which could predict in advance how the sequence produced by a particular class of generator will perform on these tests. However, with most of the pseudorandom sequence generators presented here this theoretical basis is not available. Wolfram [Wolfram1985b] suggests a more general criterion for randomness based upon computational irreducibility, but it appears that further work is required to formalise this latter

approach. His measure of randomness is that generated patterns must not be recoverable by any algorithm which is polynomial in time, thereby placing the computation of the sequences into the class of NP-complete problems [Garey1979]. Therefore, since no general mathematical techniques are available for the analysis of some of the following PRNGs one must rely on empirical tests, such as those given above, to judge the suitability of a particular PRNG.

2.2.2. Pseudorandom Sequence Testing

The criterion used here for judging whether or not a pseudorandom number generator produces pseudorandom sequences will be based on the performance of a number of different generated sequences on the random number tests described above. The results will be presented in tabular form. The tests are made on 30 bit integer sequences consisting of either 1000 or 10000 numbers. Each test is made using two moduli to exercise some tests which are not valid for all values of d. The results given in the random number test tables were obtained by running 100 different sequences through the random number tests and then tabulating the percentage of pass/fail results. It is expected that different pseudorandom sequences will not pass all the tests every time since occasionally it is expected that a truly random sequence will not pass a given test; i.e. we know that a pseudorandom sequence will occasionally generate a value of V outside the range $d \pm 2\sqrt{d}$. The percentage of times that a given test is passed by sequences generated by a certain PRNG determines the pass/fail result. The percentage required to designate whether a generator passes or fails a given random number test is not a firm number since we are forming statistics from statistics. Knuth [Knuth1981] suggests that three sequences from a generator be tested and if a majority (i.e. at least two out of three) of these sequences pass a given test then the generator is said to pass that particular test. In this work we have used many more sequences per generator to increase our confidence in the randomness of the new generators proposed in this chapter. Therefore, if Knuth's two out of three criterion is used, a 66% pass rate would be sufficient for a PRNG to pass a particular test. To make the pass/fail criterion more rigorous, the percentage should be increased. However, it cannot be made too high since we expect some pseudorandom sequences to fail some tests. The random number test tables presented in this chapter consider a PRNG to pass a particular test if it possesses a 75% pass rate.

Another metric which indicates the quality of randomness in a pseudorandom sequence is the number of tests failed by a particular sequence. We expect that some pseudorandom sequences will fail a given test, hence what is of more importance is how many tests a particular sequence fails. If a sequence passes all tests but one then this sequence must be considered pseudorandom but containing values which fail a particular test. If several tests are failed by the same sequence then it is probable that the sequence is not pseudorandom. Therefore, one should examine the worst case performance of all the sequences to see if a particular PRNG will produce a sequence which performs badly on many tests. This is also not the best metric to use since with pseudorandom sequences it is possible that a particular pseudorandom

sequence will fail a number of tests. To overcome this problem, a weighted average of the number of tests failed is proposed using the following metric

$$\frac{\sum_{j=1}^{N}\sum_{i=1}^{T} A_i \alpha_{ij}}{N \sum_{i=1}^{T} A_i} \tag{2.16}$$

where

T = number of random number tests .

N = number of sequences tested .

A_i = weighting for test i .

$$\alpha_{ij} = \begin{cases} 0 & \text{if test i passed by sequence j .} \\ 1 & \text{if test i failed by sequence j .} \end{cases} \tag{2.17}$$

This metric will describe the result of the average performance of sequences produced by a PRNG on all tests. The weighting allows some tests to have an increased importance. For example, if it were crucial that a pseudorandom sequence have equidistribution then the weighting for the equidistribution test could be made large so that it will affect the weighted average much more than other tests. Here we do not intend to apply the sequences to any given algorithm so, we will consider all A_i to be equal (i.e. all tests are evenly weighted).

The different tests are referred to by number and can be referenced against the key in Table 2.1. It should be realised that the 100 sequences used for these results is actually only a small sample of all the possible sequences. This is true for two reasons: firstly, the computer time to generate just one table of results for a 10000 number sequence takes approximately 40 hours on a SUN3-160; secondly, the number of possible sequences is very large. Here sequences consisting of 30 bit integers were tested. We know that each starting value in a deterministic sequence will generate a unique sequence, so we have 2^{30} possible sequences. Therefore, if all 100 tested sequences pass most of the random number tests then we can say with 75% confidence that less than 10% of the sequences could be nonrandom. A more complete analysis of this confidence using material adapted from [Papoulis1965] is given in Appendix A.

2.3. CONVENTIONAL TECHNIQUES

The following is a brief description of several techniques of pseudorandom number generation which have found widespread use in computer generation of pseudorandom sequences.

There are two main categories of pseudorandom sequence generators in general use: those which employ software algorithms and those which employ hardware

Test #	Test name
1	Equidistribution
2	Serial Doubles
3	Serial Triples
4	Serial Quads
5	Gap
6	Poker, k=3
7	Poker, k=4
8	Poker, k=5
9	Poker, k=6
10	Poker, k=7
11	Poker, k=8
12	Coupon
13	Permutation, q=2
14	Permutation, q=3
15	Permutation, q=4
16	Permutation, q=5
17	Permutation, q=6
18	Run up
19	Run down
20	Max of t
21	Serial correlation
22	Bit sequence correlation
23	Visual
24	Worst case fail
25	Evenly weighted average

Table 2.1: *Key to the random number test tables. The results for tests 1 to 21 are recorded as the percentage of sequences passing the test. If a test is invalid for the given moduli* **skip** *is recorded. The value given for test 22 is the value of the autocorrelation statistic C_A. If the cross-correlation between adjacent bit sequences is ≤ 10% then test 23 records* **Pass**. *However, if the cross-correlation is > 10% then* **Fail** *is recorded. Finally, tests 24 and 25 give the largest number of tests failed by a sequence and the weighted average number of tests failed by all sequences.*

techniques. Here we are concerned with high speed generation of pseudorandom numbers in VLSI, so the latter category is of primary interest. However, the algorithm based generators can be mapped to hardware and analysed using the traditional hardware measures of area and time as described in the first subsection below. This allows some techniques from the former category to be compared with the traditional hardware based generators for use in the fine-grained parallel processing architectures

under consideration.

2.3.1. Hardware Conversions of Algorithmic Techniques

2.3.1.1. Box-Muller Transformation

Many PRNGs operate on real numbers producing a sequence such as in Eqn. 2.1. An example of such a technique is the Box-Muller transformation [Box1958] which is sometimes referred to as the polar method. Here one starts with two independent random variables u and v which are both uniformly distributed on [0,1]. If the transformations

$$x = \sqrt{-2 \log u}\ \cos(2\pi v) \tag{2.18}$$

and

$$y = \sqrt{-2 \log u}\ \sin(2\pi v) \tag{2.19}$$

are employed then x and y are independent pseudorandom numbers normally distributed on [−1,+1] [Smith1985]. A real number sequence from zero to one can be created by placing the absolute values of x and y into a sequence. The next two sequence values are made by setting $u = x$ and $y = v$ and applying the transformations given above to get new values for x and y. Note that if $\sqrt{-2 \log u} > 1.0$ then x and y may be greater than 1. This can be overcome by repeatedly dividing the value of $\sqrt{-2 \log u}$ by 2 until it is less than 1.0. This technique is intended for applications which require deriving two normally distributed variables from two uniformly distributed variables. Iterative application of this technique, as described here, will then form a sequence which more closely resembles a normally distributed rather than a uniformly distributed pseudorandom number sequence. The performance of this generator on the random number tests is given in Table 2.2. Note that, despite the non-uniform distribution of the resulting sequences, the equidistribution test is still passed. This is because the acceptance range for reasonable values of V using the $d \pm 2\sqrt{d}$ metric is large enough to pass sequences whose distribution lies between uniform and normal. The results indicate that, on average, a generated sequence fails only 1.2 tests with a worst case performance of about 4. This indicates that the generated sequences would perform very satisfactorily as pseudorandom sequences with a caution on the uniform distribution of numbers.

The major problem with this technique is in the computational cost of determining the square roots, logarithms, sinusoidal functions, and the fix to ensure that x and y are in the range [−1,+1]. Even in a system with floating point hardware the time expended in generating a pseudorandom sequence can be considerable. It should be noted that the computational complexity of the Box-Muller method is comparable to other real number PRNG techniques. Therefore, for applications in which an abundance of spatially distributed pseudorandom numbers are needed, this method is computationally prohibitive. In the area of interest (parallel hardware pseudorandom number generation in a fine-grained processor network) this approach to PRNG is

dismissed since it is well known that floating point hardware requires considerable silicon area and time compared to the integer methods which follow.

2.3.1.2. Linear Congruential Generator

Another well known technique for generating pseudorandom sequences in software which is much less computationally demanding employs linear congruential generators [Knuth1981]. Here a linear congruential sequence $<X_i>$ is generated using the transformation

$$X_{n+1} = (aX_n + c) \ mod \ m, \ n \geq 0 \tag{2.20}$$

where

m = the modulus; $m > 0$.
a = the multiplier; $0 \leq a < m$.
c = the increment; $0 \leq c < m$.
X_0 = the starting value; $0 \leq X_0 < m$.

The values selected for m, a, and c determine the quality of the pseudorandom numbers generated. For implementation in hardware we require a multiplier and adder both modulo m. A consideration of the area and time requirements of this generator is given in the analysis of the multiplicative congruential pseudorandom number generator described below.

2.3.1.3. Multiplicative Congruential Generator

Often the value of c in the linear congruential generator is set to 0 which reduces Eqn. 2.20 to

$$X_{n+1} = a \ X_n \ mod \ m \ . \tag{2.21}$$

This is the standard generator available on most computer systems and is called a multiplicative congruential generator. Typical values of a and m are $a = 1664525$ and $m = 2^{32}$ which is convenient for 32 bit CPU systems or $a = 3141592653$ and $m = 2^{35}$ for a 35 bit CPU [Knuth1981]. The cycle length can be increased by using a larger modulus with the appropriate multiplier. Both the linear congruential and multiplicative congruential generators have been extensively studied and analysed [Knuth1981] for values of a, m, and c, yielding maximal-length sequences of high quality pseudorandom numbers [Fishman1982]. In Table 2.2 the random number tests as applied to a multiplicative congruential generator implemented as

$$X_{n+1} = 1664525 \ X_n \ mod \ 2^{32} \tag{2.22}$$

are presented and indicate its high quality.

While these techniques are certainly an improvement over the Box-Muller method in terms of computation, the computational expense is still considerable. Consider a generator corresponding to Eqn. 2.22. The equation must be solved for each pseudorandom number. In the case of parallel architectures one must generate a

Sequence length = 1,000

Test	Box Muller		Multiplicative Congruential		Additive Feedback	
mod	10	100	10	100	10	100
1	93	97	91	96	93	91
2	89	89	93	94	94	86
3	89	skip	93	skip	94	skip
4	86	skip	90	skip	88	skip
5	95	99	97	93	93	95
6	96	93	94	96	94	95
7	93	97	93	96	95	98
8	96	98	95	100	91	96
9	94	98	94	97	99	96
10	93	97	94	93	97	96
11	97	99	98	96	95	99
12	92	skip	92	skip	93	skip
13	98	99	98	99	98	97
14	95	94	94	92	97	96
15	skip	89	skip	92	skip	90
16	skip	88	skip	90	skip	93
17	skip	91	skip	81	skip	98
18	skip	94	skip	94	skip	91
19	skip	94	skip	91	skip	91
20	94	89	89	93	88	81
21	75	66	78	65	70	70
22	0.08		0.09		0.12	
23	Pass		Pass		Pass	
24	5	6	7	5	6	5
25	1.25	1.29	1.17	1.42	1.21	1.41

Sequence length = 10,000

Test	Box Muller		Multiplicative Congruential		Additive Feedback	
mod	10	100	10	100	10	100
1	96	93	96	95	90	91
2	97	85	89	93	93	95
3	94	skip	92	skip	90	skip
4	91	skip	97	skip	89	skip
5	96	93	97	93	93	94
6	95	97	96	97	96	96
7	91	99	96	98	91	98
8	93	96	98	97	92	92
9	91	96	96	98	93	100
10	99	92	93	95	95	97
11	91	98	96	98	97	98
12	93	skip	90	skip	96	skip
13	99	94	98	99	93	95
14	95	95	96	95	91	93
15	skip	87	skip	86	skip	91
16	skip	90	skip	90	skip	95
17	skip	87	skip	88	skip	88
18	skip	89	skip	88	skip	89
19	skip	88	skip	92	skip	88
20	94	89	93	91	92	88
21	67	74	67	66	63	68
22	0.03		0.03		0.03	
23	Pass		Pass		Pass	
24	4	5	5	4	5	7
25	1.18	1.58	1.10	1.41	1.46	1.44

Table 2.2: *Three algorithmic pseudorandom number generators and their test results. The moduli* 10 *and* 100 *were used for the random number tests.*

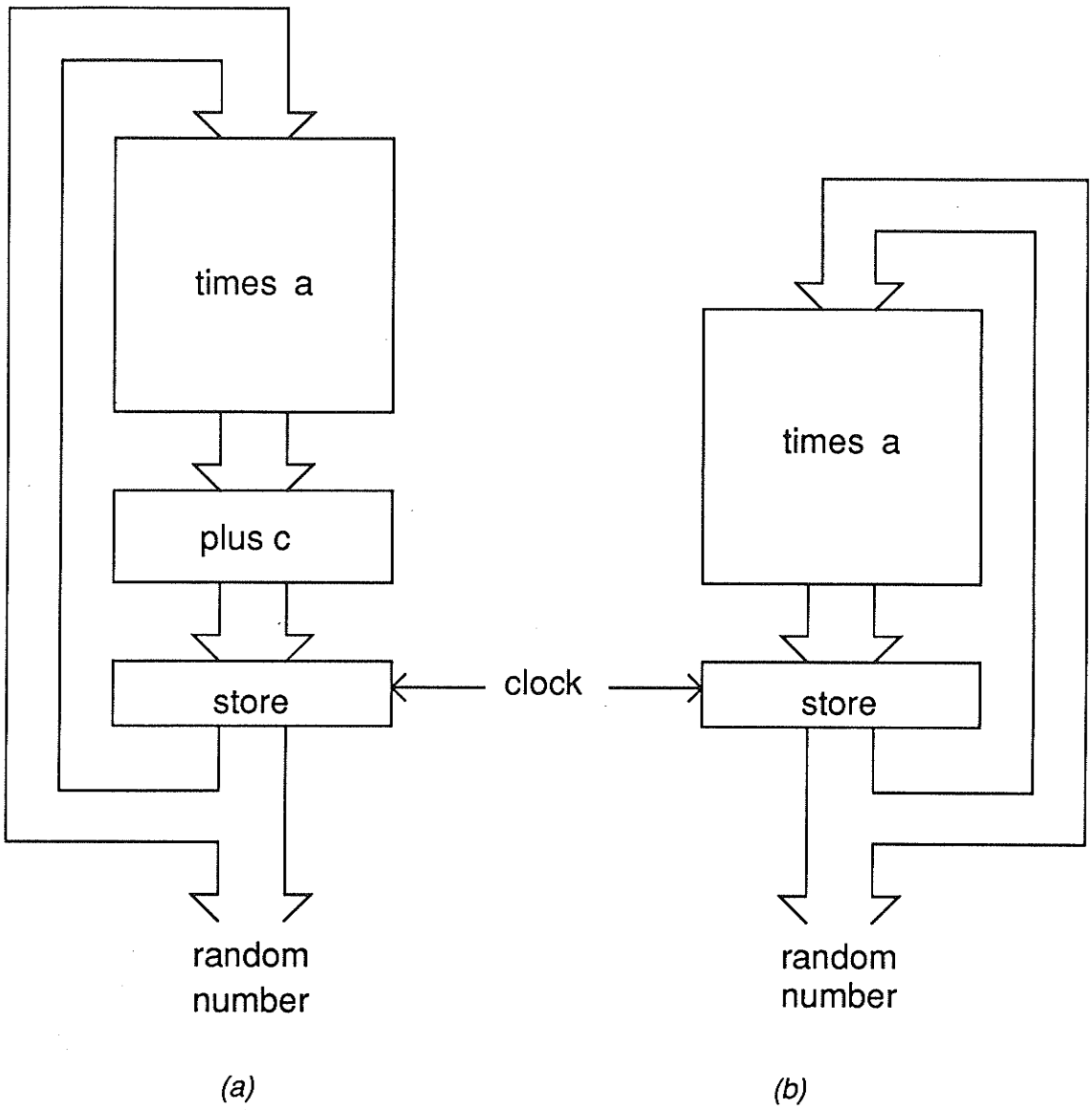


Figure 2.3 : *A hardware implementation of (a) a linear congruential generator and (b) a multiplicative congruential generator.*

pseudorandom sequence at each separate processor site, both rapidly and economically in terms of area. The required hardware consists of a modulo m multiplier and some registers, as shown in Fig. 2.3, both for the linear congruential generator (Fig. 2.3(a)) and the multiplicative congruential generator (Fig. 2.3(b)). The two methods use very similar hardware since the only difference is the addition of an increment value in the linear congruential generator. For analysis only the multiplicative generator will be considered. One would presumably set the value of m equal to the word size of the processors (we would not expect the word size of processing elements in a fine-grained parallel processing architecture to be 32 bits) but care must be

taken since the value of m affects the cycle length of the pseudorandom sequence and the value of the multipler, a. Nevertheless, the PRNG will require at least the area of an integer multiplier at each processing site. The area used by a hardware 8 bit multiplicative congruential generator is $6.0 \times 10^6\ \mu m^2$.

The other option is to use only one PRNG for all the processors. One then experiences a degraded time performance in the generation of the pseudorandom numbers, since for n processors it will require at least n clock cycles to generate a new pseudorandom number for each processor. This is especially evident in computations where many pseudorandom numbers are required, such as in Monte Carlo simulations. The area-time, or AT, metric in both cases is the same, but again a more efficient technique of generating the pseudorandom numbers is desired.

A final technique which may be used, is to generate a large global pseudorandom number using a single generator and then supply each processor with its own local pseudorandom number by using selected bits from the global pseudorandom number. While this method will provide pseudorandom numbers to each processor in one time step, there are a number of problems associated with this technique. The primary problem is that each processor requires a pseudorandom number which is not correlated with pseudorandom numbers at other processors. Therefore, the global pseudorandom number must have a word size equal to the number of processors times the local word size. This leads to an extremely large word size thereby making its use prohibitive since the area and time measures of the multiplier both scale with the square of the word size. A lesser problem is the routing of the local pseudorandom numbers to their respective processors which may use considerable additional area to that of the PRNG.

2.3.1.4. Additive Feedback Generators

A considerably more efficient technique, in terms of the silicon area requirements, is based on an additive feedback PRNG [Tausworthe1965], [Golomb1982]. A general pseudorandom sequence generator combines past numbers in the sequence to produce a new number. Consider the sequence

$$X_n = (a_1 X_{n-1} + a_2 X_{n-2} + \cdots + a_k X_{n-k}) \bmod m \tag{2.23}$$

where each X_i is an ordered element of the sequence, a_i is a multiplier for past elements of the sequence, and m is the modulus of the sequence (usually the word size of the computer). The value of k determines how many past sequence values must be stored. If m is prime and

$$y^k - a_1 y^{k-1} - \cdots - a_{k-1} y - a_k \tag{2.24}$$

is an irreducible polynomial over GF(m) then the sequence defined by Eqn. 2.23 produces a sequence of length $m^k - 1$.

Consider a binary sequence given by

$$x_n = a_1 x_{n-1} + a_2 x_{n-2} + \cdots + a_p x_{n-p} \bmod 2\ , \tag{2.25}$$

where x_i is the i'th bit in the sequence. The resulting binary sequence is pseudorandom with a maximum cycle length of $2^p - 1$ if, and only if, the polynomial

$$y^p + a_1 y^{p-1} + a_2 y^{p-2} + \cdots + a_{p-1} y + a_p \tag{2.26}$$

is primitive over GF(2) [Golomb1982]. We can form m bit pseudorandom words $<X_i>$ by combining m such binary sequences in parallel provided there is no correlation between binary sequences x_i and x_j. Therefore, a sequence of m bit words defined as

$$X_n = a_1 X_{n-1} + a_2 X_{n-2} + \cdots + a_p X_{n-p} \tag{2.27}$$

where each addition corresponds to adding bit streams without carry, will generate sequences of cycle length $2^p - 1$. Further computational advantages can be realised by using primitive polynomials which contain only two or three terms. Primitive polynomials of the form $1 + a^q + a^p$, $p > q$ have been tabulated up to a large order [Zierler1969]. Also, bitwise addition without carry is equivalent to the *exclusive–or* operation, denoted as $\oplus$. Therefore, using a primitive trinomial and the *exclusive–or* operation, Eqn. 2.27 reduces to

$$X_n = X_{n-q} \oplus X_{n-p} \quad . \tag{2.28}$$

This requires only one m bit parallel *exclusive–or* operation for each new pseudorandom number. It has been shown that small values of q or q near $\frac{p-1}{2}$ should not be used due to bad run properties [Tootill1971]. However, if p and q are carefully chosen then good pseudorandom number properties will result [Whittlesey1968]. A popular pseudorandom sequence generator of this type is the so called R250 pseudorandom sequence generator where the values of $p = 250$ and $q = 103$ are used yielding a sequence of length $2^{250} - 1$ [Kirkpatrick1981]. Therefore, a pseudorandom number generator based on these principles will usually be much faster than multiplicative congruential generators and will still deliver similar performance on the random number tests, as shown in Table 2.2. However, unlike the multiplicative congruential generator where proper selection of the multiplier, a, and modulus, m, guarantees a good pseudorandom sequence with little consideration for the starting value X_0 (except of course for $X_0 = 0$), the additive feedback generator must be carefully initialised. If the i'th and j'th bits are the same in each of the first p numbers of the sequence (i.e. the seed values) then they will remain the same over the entire sequence. Also, if the i'th and j'th bits only differ slightly then it will require many iterations before bits i and j become independent. This problem is the subject of several papers and some techniques to produce good seed values have been described [Kirkpatrick1981], [Lewis1973].

For the parallel processing applications under consideration here, the implementation of this generator in hardware must again be considered since the time spent computing the pseudorandom numbers at each processor may dominate the computation and memory requirements of each processor. A possible configuration is given in Fig. 2.4. This generator will use considerably less circuitry than the multiplicative

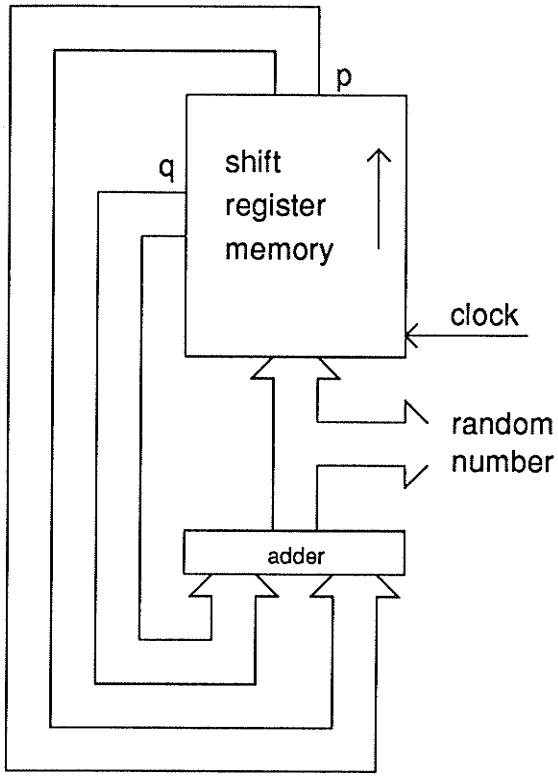


Figure 2.4 : *A hardware implementation of an additive feedback generator.*

congruential generator at each processing site since all that is required is a memory of p numbers and a word wide *exclusive–or*. Note that the memory actually consists of m p-bit shift registers since only the p most recent numbers in the sequence are retained and the position of past numbers in the memory increases by one for each new pseudorandom number. The alternative is to use a general purpose memory with addressing circuitry which will determine the actual location of the numbers corresponding to X_{n-q} and X_{n-p} and the next available location in the memory for the new X_n. However, this will complicate the control unnecessarily since the shift register implementation will automatically place the memory contents in the correct location. This collection of shift registers will use the overwhelming majority of the area of this implementation. For example, if the R250 generator is implemented, we require m 250 bit shift registers at each processing site solely for the purpose of generating the pseudorandom sequence. This area can be substantial when compared to the area required by the actual computation which occurs at the processing site. One way to reduce this area is to use a smaller value for p. However, the value of p affects several aspects of the generator such as the cycle length, $2^p - 1$, and the correlation of different bit streams so the value of p should be changed with care. Even if the value of p can be reduced to a much smaller value such as 30 (cycle length of $2^{30} - 1$) then m 30 bit shift registers are required which remains a very large area when compared to some of the techniques for pseudorandom number generation

which follow. The area used by an 8 bit additive feedback generator when $p = 30$ is $3.5 \times 10^6\ \mu m^2$. Another point to note is a communication problem since we must feed the p'th and q'th numbers in the shift register memory back to the *exclusive–or* gates at the beginning of the shift registers. This leads to the conclusion that, in general, additive feedback PRNGs are not satisfactory for fine-grained parallel processing requiring pseudorandom numbers.

A modification to the additive feedback generator is developed in [Pearson1983b]. Here a generator based on shift register sequences is proposed. It has the desirable properties of high speed and small area in that a pseudorandom number can be produced every clock cycle and no large data memory is required. This PRNG is targeted towards implementation using VLSI technology but to the author's knowledge no such implementation has been reported. In comparing this scheme to other PRNGs we see while there is no data memory required it utilises a general feedback shift register structure and so requires several shift registers per bit of random number. In addition, many *exclusive-or* gates are required. Therefore, this particular PRNG was not more fully investigated since the area used is still larger than that for some of the pseudorandom number generation techniques which follow.

2.3.2. Hardware Techniques

2.3.2.1. True Random Number Generators

In hardware some truly random sequence generators can be built. For example, if we count the number of particles emitted from various radioactive sources over a short period of time, a sequence of truly random numbers is produced. Another similar technique is to observe thermal noise from a device such as a resistor. In fact, observation of any source of white noise will lead to a truly random sequence. The hardware required to observe, and thereby determine the actual random number sequence, exists but it certainly would not, in the present context, provide an efficient technique for generating random numbers. We seek to generate many random numbers in parallel, so the conventional noise-based methods, while certainly giving truly random sequences, cannot be considered appropriate.

2.3.2.2. A Shift Register Based RNG

A technique to generate random numbers using nondeterministic methods but which can be implemented in a VLSI circuit without requiring an unreasonably large area is given in [Letham1986]. Here a shift register is used in conjunction with three free-running ring oscillators as in Fig. 2.5. As the temperature of the integrated circuit varies, the operating frequency of the three ring oscillators changes. A local *silicon heater* is placed around one of the fast ring oscillators to vary the difference in speed of the two fast ring oscillators. The outputs from the two fast ring oscillators are *exclusive–or*ed and then sampled by the slow ring oscillator. The clock of the shift register will be asynchronous to the speed of the ring oscillators, so the bit stream

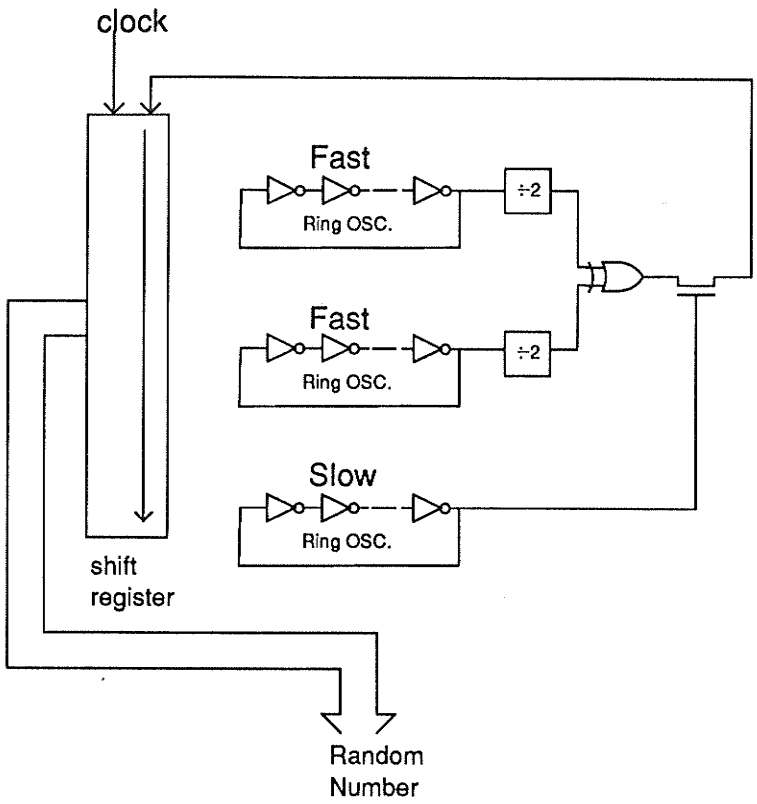


Figure 2.5 : *Shift register based random number generator.*

being clocked into the shift register will be random. In addition, the shift register is completely shifted a *random* number of times based on the value of several bit positions in the shift register. This process causes the contents of the shift register to be completely unpredictable. The time in clock cycles required to generate a random number is at least the length of the shift register since for each new random number new values must be clocked into each location of the register. Therefore, for an m bit number a multiple of m time steps will occur before a new random number is available. The process of generating these numbers is not deterministic since nondeterministic factors are contributing to the numbers that are being produced. However, observation has shown that while the sequence is completely unpredictable, it tends to produce a few values much more often than would be expected of a random sequence [Letham1986]. This makes this type of PRNG difficult to use in algorithms where the distribution of random numbers is assumed to be uniform or white. Favourite numbers can cause systematic weaknesses in nondeterministic algorithms. Testing by standard random numbers tests, other than simple correlation tests, has not been performed on the generated sequences, so it is possible that higher correlation and run tests may show problems with these sequences. This makes this PRNG unsuitable for our purposes since it may be suffering from Knuth's observation that random numbers should not be generated with a method chosen at random [Knuth1981].[2.4]

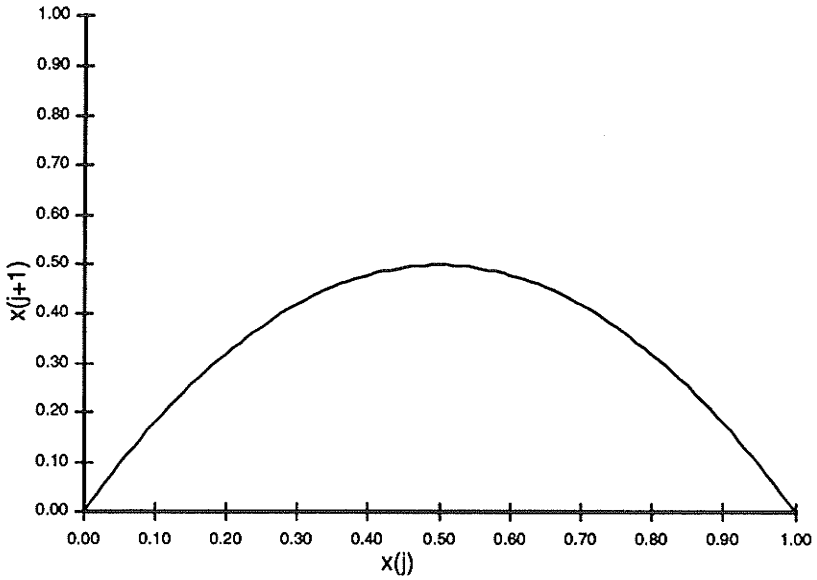


Figure 2.6 : *Transfer function of Eqn. 2.29. Here* $\lambda = 0.5$.

The estimated area of this approach is $0.44 \times 10^6\ \mu m^2$ for an 8 bit RNG. The circuitry used, while smaller than that required to implement any of the other true RNGs, uses more area than the techniques which follow. It may cause further difficulties in implementation if several of these RNGs were used in parallel on the same chip, because of the required local heating of the silicon surrounding the ring oscillators.

2.3.2.3. Using Chaos as a RNG

Some simple deterministic systems can exhibit a property which has become known as chaos [May1976]. One of the fundamental properties of this phenomenon is unpredictability of the output from a chaotic circuit. Several nonlinear circuits which have been observed to display chaotic behaviour have been recently reported [Rodrigues1986]. It has also been recently proposed to use these circuits as noise generators [McGonigal1987]. The description given in this section is not intended to describe in detail how chaos is created in a simple nonlinear circuit. Rather, its purpose is to show the feasibility of such circuits as PRNGs for parallel computing.

One of the most common ways of creating chaos is to use the logistic map [May1976]

$$x_{j+1} = 4\lambda x_j(1 - x_j);\ j = 0, 1, 2, \cdots;\ 0 < x_0 < 1 \quad . \tag{2.29}$$

If we plot the output versus the input (i.e. x_{j+1} versus x_j) as in Fig. 2.6, we get a

[2.4] If further investigation of this approach indicates that it passes standard random number tests and the distribution problems are solved this technique may prove to be satisfactory for single number at a time applications.

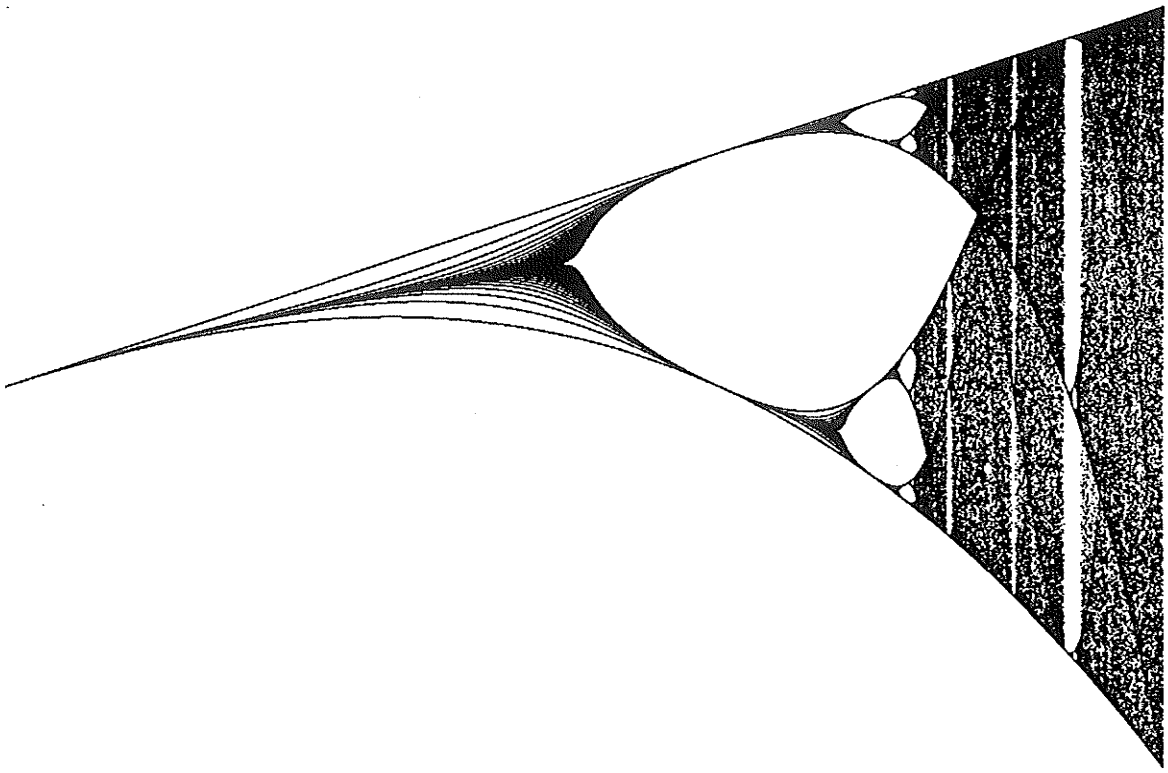


Figure 2.7 : *Output of Eqn. 2.29 versus* λ *for* $0.50 < \lambda \leq 1.0$. *For each value of* λ *100 output values are plotted.*

parabolic transfer function of height λ. It is known that for λ between 0.50 and 0.89 the value of x_j oscillates periodically between 2^n states, where $n = 1, 2, 3, \cdots$ (depending on the value of λ). For $0.89 < \lambda < 1.0$ the oscillation has no detectable period and is considered to be chaotic. In Fig. 2.7 a plot of the output versus λ is shown for $0.75 < \lambda \leq 1.0$. Note that we observe the well known period-doubling route to chaos [May1976]. It is known that the power density spectrum of the output is white [Grossmann1977] and so, the output values must be equidistributed. In [McGonigal1987] the circuit of Fig. 2.8 is proposed which iterates the logistic map and so could be used as a RNG. This circuit requires several analogue components including differential amplifiers, gain amplifiers, and switching capacitors.

While it is possible to implement such circuits using an MOS process, it is obvious that requiring analogue components places some restrictions on the feasibility of combining digital computing hardware with chaotic circuits in a VLSI environment. Also, as with the previous RNG, while the output of a chaotic circuit is certainly unpredictable, further tests are required to verify its suitability as a PRNG. Finally, such a circuit is not small and to get large word sizes will require either an analogue to digital converter (ADC) with resolution equal to the required word size, or else a unique chaotic circuit corresponding to each bit position in the random word. Therefore, the area will be quite large if the random word is to appear in parallel. If a serial

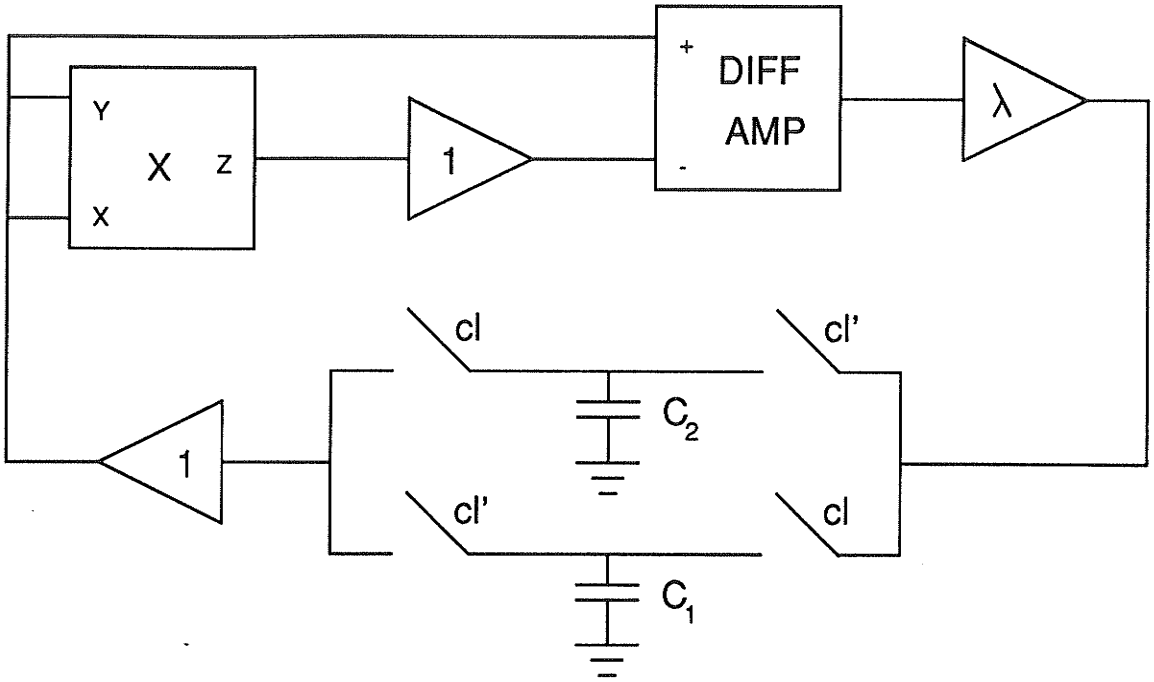


Figure 2.8 : *A MOS realisable circuit implementing the logistic map. Taken from [McGonigal1987].*

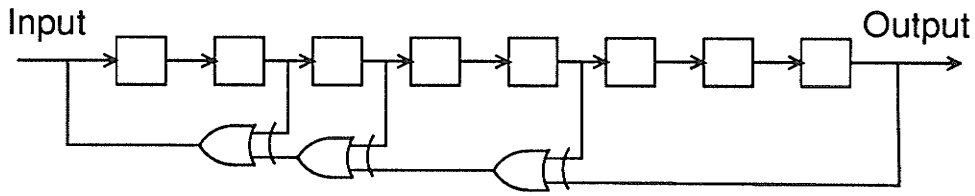


Figure 2.9 : *An 8 bit linear feedback shift register implementing the polynomial* $x^8 + x^5 + x^3 + x^2 + 1$.

accumulation scheme where the output from one chaotic circuit is sampled and accumulated is used then the time to acquire a new random number could be excessive. Thus, for the purposes of this work we will ignore chaotic circuits but it is possible that some chaotic circuits may be suitable RNGs for some applications.

2.3.2.4. Linear Feedback Shift Register

The most popular hardware pseudorandom sequence generator is the linear feedback shift register (LFSR). Figure 2.9 shows a circuit diagram for an 8 bit LFSR. The binary sequence at bit *i* is generally considered to display attributes of a

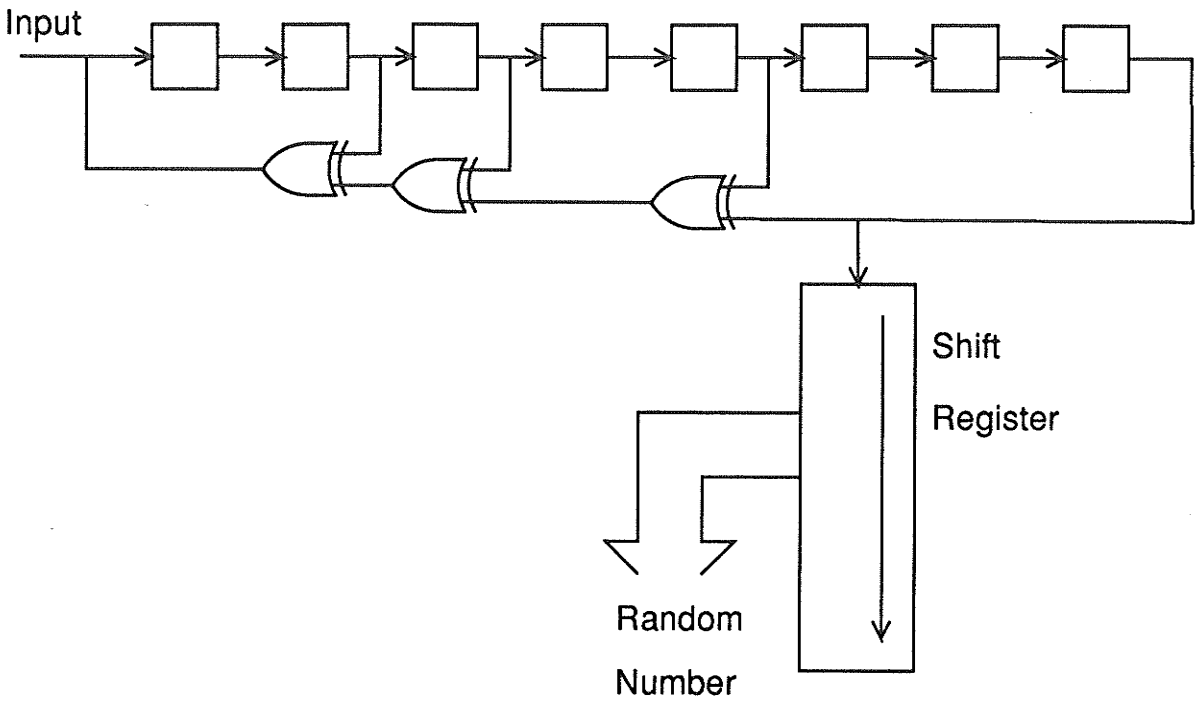


Figure 2.10 : *Forming random words from a single bit in the LFSR using a serial-in/parallel-out shift register.*

pseudorandom binary sequence with cycle length $2^n - 1$ for an n bit shift register, provided the polynomial describing the register is primitive over GF(2) [Golomb1982]. Note the similarity between the additive feedback generator and the LFSR. In fact, the additive feedback generator is a result of research done on the LFSR. An m bit pseudorandom number or word can be generated by collecting m bits in sequence from bit i using a serial-in/parallel-out register as in Fig. 2.10. This means that we take m clock shifts or clock cycles to form the pseudorandom word. Note that succeeding words cannot be formed on each clock cycle by shifting out the oldest bit from time $t - m$ and shifting in the new bit from time t because the succeeding numbers would be strongly correlated. To overcome this time delay in producing new pseudorandom words the bits of the LFSR are sometimes used in parallel so that a new pseudorandom word is formed on each clock cycle. A variation on this parallel technique is to wait n clock cycles between numbers from the PRNG. This PRNG will be referred to as the parallel LFSR with $\beta = n$ method. Table 2.3 compares the results of the random number tests on these three methods of pseudorandom sequence generation.

The test results for the serial-in/parallel-out method of Fig. 2.10 and the parallel LFSR with $\beta = n$ yield similar results and can be classified as pseudorandom sequences based on these results. However, the parallel LFSR method does not compare favourably with the above generators since it consistently fails almost all of the random number tests. The only test on which the parallel LFSR generator performs well is the equidistribution test. This should not be surprising since the initial value in

Sequence length = 1,000

Test	Sin Pout LFSR		P LFSR $\beta = n$		P LFSR	
mod	10	100	10	100	10	100
1	100	74	100	100	75	100
2	100	74	100	100	0	0
3	100	skip	69	skip	0	skip
4	100	skip	100	skip	0	skip
5	71	100	100	79	100	100
6	100	80	100	100	25	48
7	100	100	100	100	30	78
8	75	100	100	100	70	25
9	100	100	100	100	100	78
10	80	100	100	100	70	25
11	100	100	100	100	70	25
12	100	skip	100	skip	77	skip
13	100	100	100	100	100	48
14	100	100	100	100	78	30
15	skip	71	skip	78	skip	25
16	skip	75	skip	100	skip	25
17	skip	100	skip	74	skip	0
18	skip	100	skip	100	skip	70
19	skip	100	skip	100	skip	55
20	75	74	79	100	25	52
21	71	29	69	79	0	0
22	0.05		0.05		1.0	
23	Pass		Pass		Fail	
24	2	4	2	2	11	16
25	1.28	2.23	0.83	0.90	9.80	12.16

Sequence length = 10,000

Test	Sin Pout LFSR		P LFSR $\beta = n$		P LFSR	
mod	10	100	10	100	10	100
1	100	70	100	100	100	100
2	100	100	100	100	0	0
3	100	skip	100	skip	0	skip
4	100	skip	100	skip	0	skip
5	100	100	100	100	100	100
6	100	75	100	100	0	0
7	100	100	100	100	0	0
8	100	75	100	100	0	0
9	100	100	100	100	0	0
10	100	100	100	100	0	0
11	100	100	100	100	0	0
12	100	skip	100	skip	100	skip
13	77	75	100	100	100	100
14	100	100	100	51	0	0
15	skip	75	skip	100	skip	0
16	skip	100	skip	100	skip	0
17	skip	78	skip	100	skip	0
18	skip	70	skip	100	skip	0
19	skip	100	skip	76	skip	0
20	100	100	76	100	0	0
21	78	77	20	80	0	0
22	0.05		0.05		1.0	
23	Pass		Pass		Fail	
24	1	4	2	2	14	17
25	0.45	2.05	1.04	0.93	14.00	17.00

Table 2.3: *Random number test results for the serial-in/parallel-out LFSR (Sin Pout LFSR), parallel LFSR waiting n clocks (P LFSR $\beta = n$), and the parallel LFSR with no wait cycles(P LFSR).*

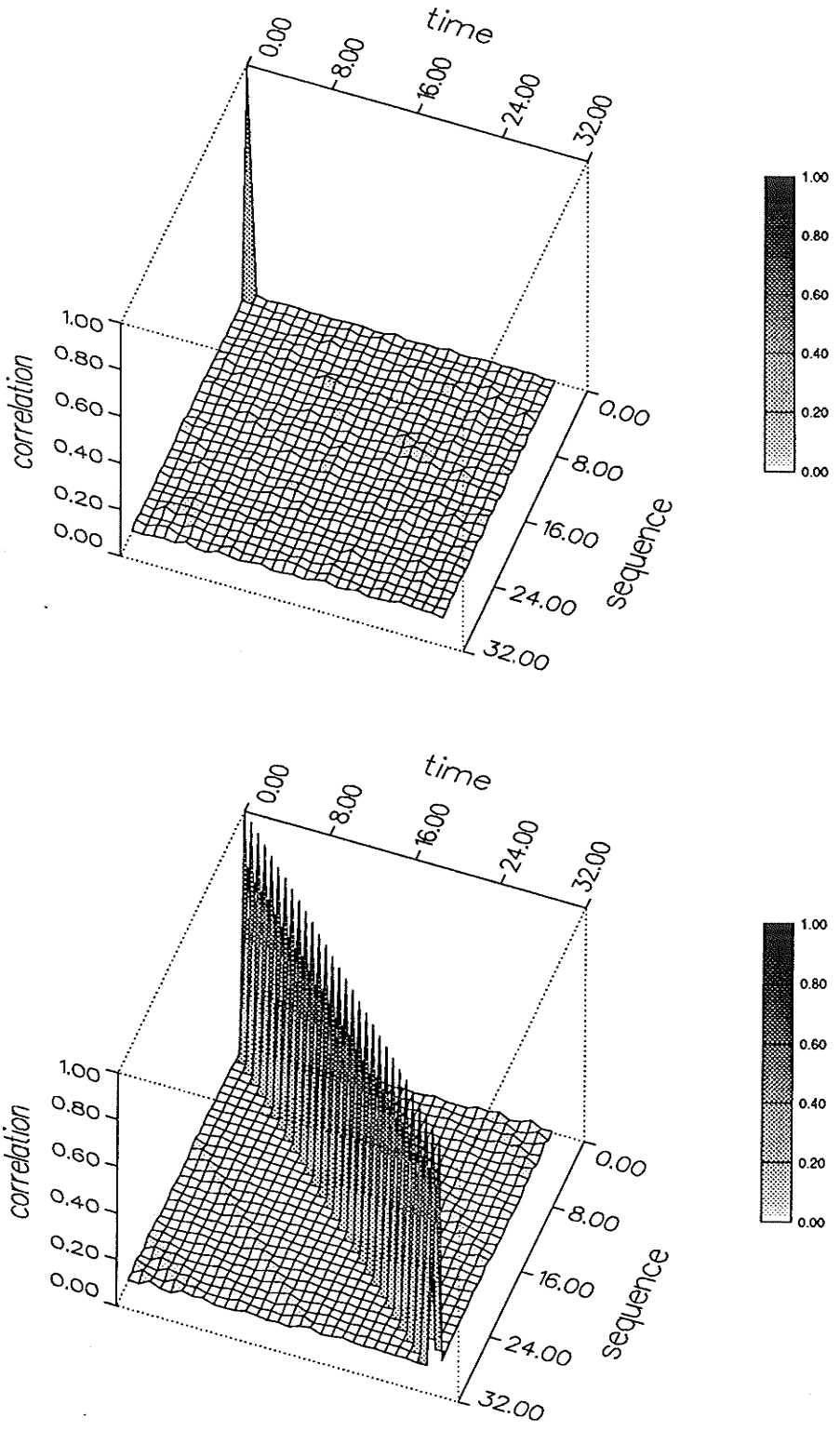


Figure 2.11 : *The cross-correlation of bit sequences in (top) the serial-in/parallel-out method and (bottom) the parallel LFSR method.*

the LFSR is continually divided by 2 and added to some value (either zero or 2^{n-1}, n = the length of the LFSR). Thereby causing the sequence of numbers to be well distributed but very predictable. This is reflected in the failure of the parallel LFSR method to pass the other tests.

Perhaps the most evident failure of the parallel LFSR is in the bit sequence correlation test. In this work correlation figures are used to show both the auto and cross-correlation of binary sequences made from 30 bit word size pseudorandom sequences. The vertical axis is the magnitude of the correlation while the x and y axes give the time displacement and sequence displacement from the reference time and sequence, respectively. To determine the autocorrelation read the values that run parallel to the time axis. For cross-correlation, read values parallel to the sequence axis till the desired sequence displacement is reached, then move parallel to the time axis to find the desired time displacement. Figure 2.11(top) shows the auto and cross-correlation of a pseudorandom sequence produced by the serial-in/parallel-out method of Fig. 2.10 and Fig. 2.11(bottom) shows the auto and cross-correlation for a sequence produced by the parallel LFSR method. Notice that the serial-in/parallel-out method yields the expected result from a pseudorandom sequence (i.e. all correlations well under 10%) while the parallel LFSR method displays a severe correlation problem. In fact, the bits in the bit streams are perfectly correlated in that the value at bit i will appear at bit $j > i$ at time $t + (j - i)$. Therefore, while the values generated by considering each of the individual bits or sites in the LFSR appear to be pseudorandom, we cannot consider the register word sequences to be pseudorandom since the bits from separate LFSR sites are fully correlated. In Fig. 2.11(bottom) also notice that even away from the cross-correlation ridge, the correlation is very regular in a wavelike pattern.

To generate pseudorandom numbers in parallel, both the LFSR and the associated register from Fig. 2.10 should be placed at each processing site. If we form the pseudorandom numbers using the scheme of Fig. 2.10 good pseudorandom numbers are formed every m clock cycles. The area for an 8 bit LFSR PRNG as in Fig. 2.10 is $0.26 \times 10^6\ \mu m^2$. In general, more than $2^5 - 1$ pseudorandom numbers will be needed (note that each 8 bit number requires 2^3 new bits from the LFSR) so, a more realistic PRNG using LFSRs to consider is a 32 bit LFSR with one output feeding an 8 bit serial-in/parallel-out register. This produces $2^{29} - 1$ pseudorandom numbers and uses $0.65 \times 10^6\ \mu m^2$. Alternatively, we could use the parallel LFSR with $\beta = n$ and wait n clock cycles between numbers to completely shift the old number out of the LFSR. When used with processors which require more than n clock cycles between pseudorandom numbers, the problem of waiting n clock cycles for a pseudorandom number is unimportant since it will not affect processor throughput. However, this will not generally be the case in fine-grained processor arrays. In both cases the processor will be forced to wait a fixed number of clock cycles before the new pseudorandom number is generated. Comparing the AT metrics for both methods generating 8 bit pseudorandom numbers it can be seen from Fig. 2.12 that the parallel LFSR approach loses its advantages after the length of the LFSR becomes ≥ 8. This is because the

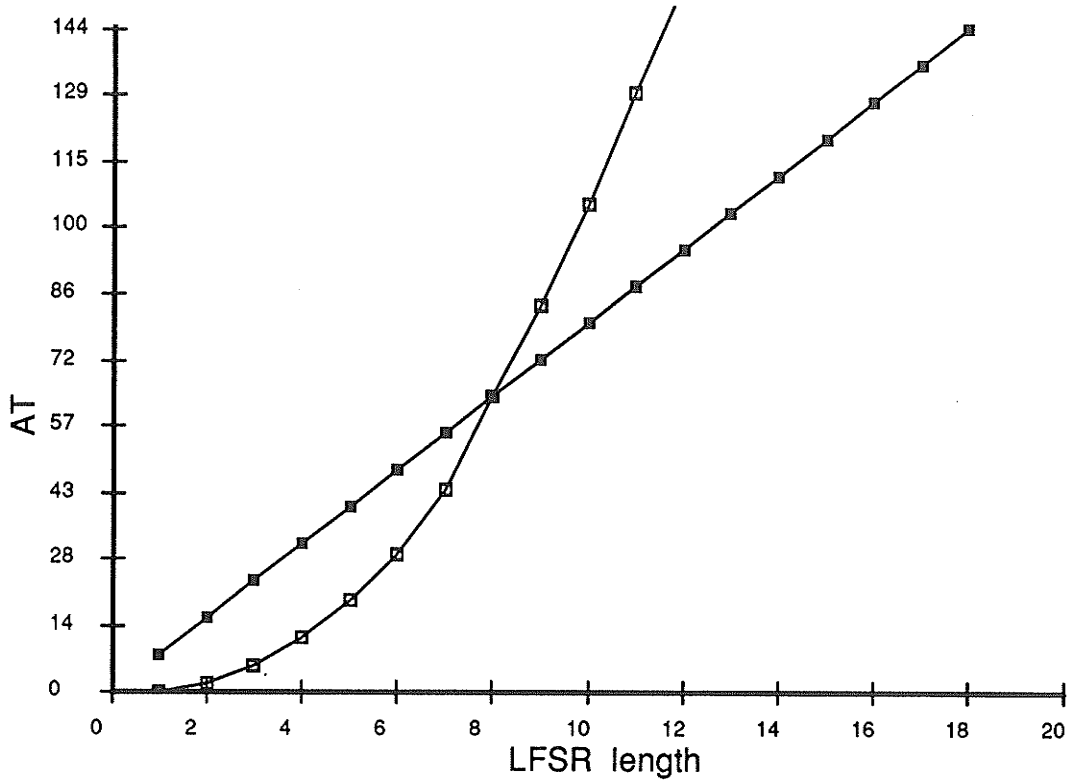


Figure 2.12 : *AT metric for pseudorandom number generators of Figs. 2.9 and 2.10; parallel LFSR (empty squares); serial-in/parallel-out method (filled squares).*

area of the two techniques scales with n at the same rate (i.e. relative sizes remain the same) but the serial-in/parallel-out method uses constant time (e.g. 8 time steps for 8 bit numbers) while the time delay for the parallel LFSR method scales with n. The crossover value scales with the required word size of the PRNG. In the following discussions the serial-in/parallel-out method will be used because it has a better AT measure for large word sizes. However, if an application uses a sufficiently small LFSR and the processor can wait n cycles between pseudorandom numbers then the parallel LFSR with $\beta = n$ could be used.

2.4. TECHNIQUES BASED UPON CELLULAR AUTOMATA

As shown above, there are many ways of generating good pseudorandom sequences. However, when we consider fine-grained parallel processing it is not convenient to employ many of the conventional techniques because of excessive area and computation time requirements. In fact, some techniques such as the Box-Muller, linear congruential generators, and additive feedback generators require so much area at each processor site (assuming we want independent PRNGs) that it would be difficult to maintain a fine-grained approach. On the other hand, if the processors

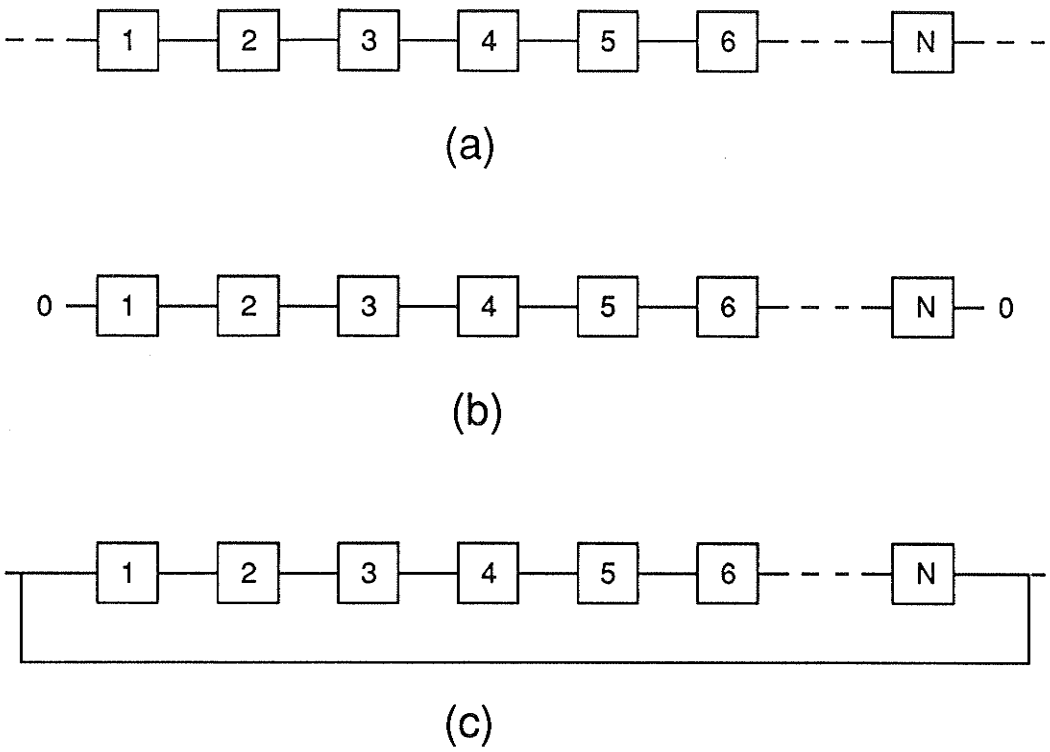


Figure 2.13 : *(a) A simple N-bit one-dimensional cellular automaton. (b) Null boundary conditions. (c) Cyclic boundary conditions.*

111	110	101	100	011	010	001	000
0	1	0	1	1	0	1	0

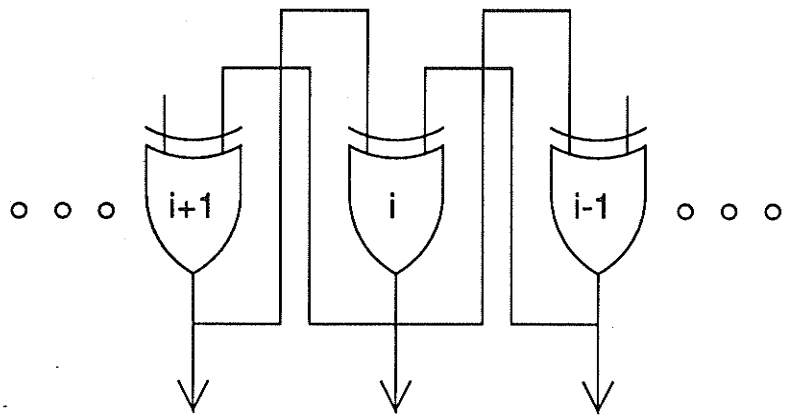


Figure 2.14 : *A rule 90 cellular automaton*

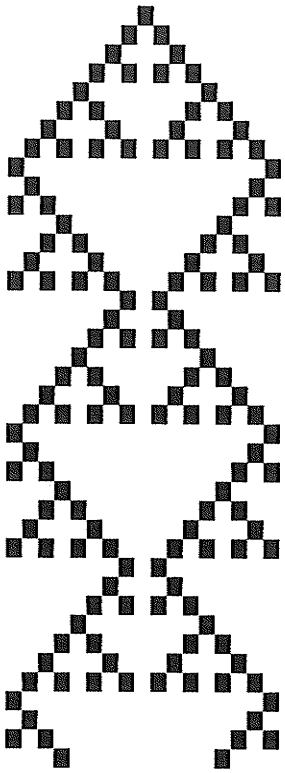


Figure 2.15 : *40 time steps in the state - time diagram for an 18 site rule 90 cellular automaton. Cyclic boundary conditions; initialised with a single nonzero site.*

cooperate in the production of the pseudorandom numbers by using a distributed PRNG then the communication overhead could dominate the computation by the processors. This leaves the LFSR as the only candidate among the conventional generators with feasible resource (area, time) requirements. However, despite the fact that the size of the LFSR lends itself to use in a fine-grained parallel processing environment, either the poor quality of the pseudorandom numbers produced or the need to wait a fixed number of clock cycles between numbers is a serious limitation. One would prefer to have new pseudorandom numbers available on each clock cycle as in an additive feedback generator, while retaining the size advantages of the LFSR. The solution proposed in this thesis is to use elementary one-dimensional cellular automata as parallel pseudorandom sequence generators for fine-grained processor arrays.

2.4.1. Introduction to Cellular Automata

The concept of cellular automata was first proposed by von Neumann [vonNeumann1963]. More recently Wolfram [Wolfram1983], [Wolfram1984b] has done much to cause a resurgence of interest in cellular automata. Most of this renewed interest in cellular automata has been kindled by the discovery that many physical problems can be mapped to these devices [Vichniac1984], [Salem1986], [Kinzel1985]. A cellular

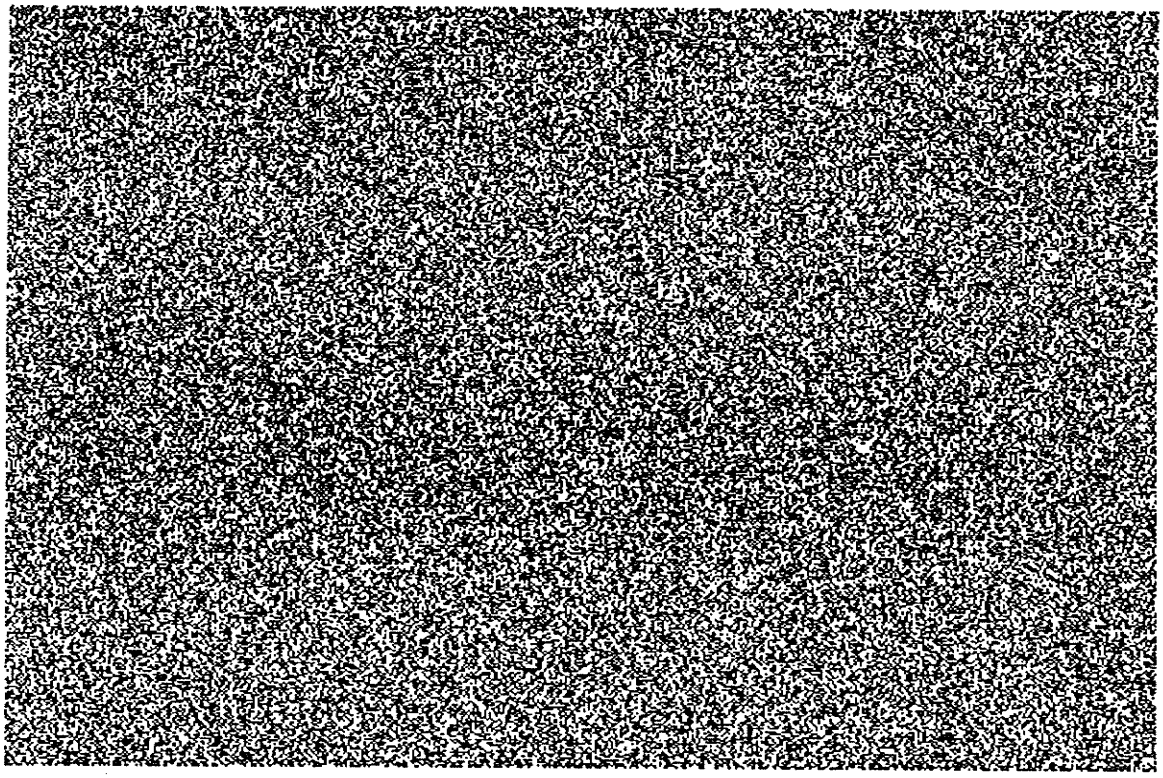


Figure 2.16 : *The raster scan output of a single site in a 32 site rule 30 cellular automaton.*

automaton evolves in discrete steps with the next value of one site determined by its previous value and that of a set of sites called the neighbour sites. The extent of the neighbourhood can vary depending, among other factors, upon the dimensionality of the cellular automata under consideration. Figure 2.13 illustrates a simple one-dimensional cellular automaton, where the next value at a site depends only on its present value and the values of the left and right neighbours. A cellular automaton may possess null boundary conditions, as in Fig. 2.13(b) (i.e. the first and last sites consider their missing neighbour site to always have a zero value), or may be cyclically connected as in Fig. 2.13(c), (i.e. one assumes the cellular automaton to form a ring thereby making the first and last sites neighbours). Here only binary one-dimensional cellular automata with two neighbour sites (left and right) will be considered, but it is possible to use any desired modulus, dimension, or neighbour set. For a binary cellular automaton of this type each site must determine its next value on the basis of the eight possible present values of itself, and the left and right neighbours (i.e. 000, 001, 010, etc...). The next state values corresponding to each possible input form a number which is referred to as the rule number under the classification scheme of Wolfram [Wolfram1983]. For example, Fig. 2.14 illustrates CA rule 90 in which the next value of a site is the sum modulo 2 of its neighbouring sites. Often the evolution of a cellular automaton is shown using a state - time diagram as in Fig. 2.15

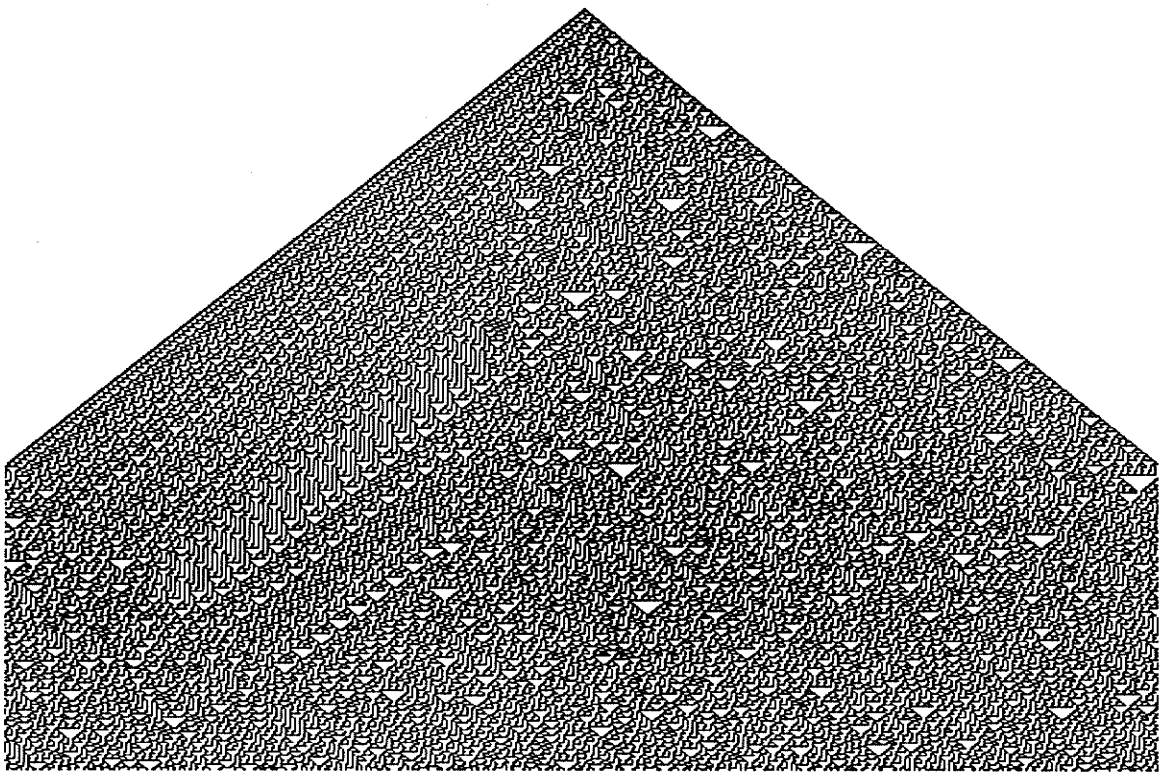


Figure 2.17 : *420 time steps in the state - time diagram of a 500 site rule 30 cellular automaton. Cyclic boundary conditions; initialised with a single nonzero site.*

where 40 time steps from the evolution of an 18 site rule 90 cellular automaton, is shown. There are, in general, two distinct methods of initialising a cellular automaton. One method is to begin with a simple state such as a nonzero value at a single central site; the other method is to begin with each site randomly initialised to 0 or 1 with $p(0) = p(1) = 0.5$. Figure 2.15 was initialised with a single nonzero site.

While the general description of one-dimensional cellular automata is very simple, different CA rules are capable of very wide ranging global behaviour. Wolfram has formulated four basic classes of behaviour in one-dimensional cellular automata [Wolfram1984c]. Class 1 automata evolve to homogeneous final global states, class 2 to periodic structures, class 3 exhibit chaotic behaviour, and class 4 yield complicated localised and propagating structures. The first two classifications are readily predictable and show little or no properties of interest for pseudorandom number generation. The third class yields much more complex behaviour in that the detailed patterns can no longer be predicted (it may still be possible to make statements about global behaviour) and often seem random in nature. Wolfram considers class 3 CA rules to be an abstract model of randomness in nature and thus very suitable for pseudorandom number generation [Wolfram1985b]. This is because the cumulative effect of many iterations in a number of class 3 CA rules is equivalent to performing very

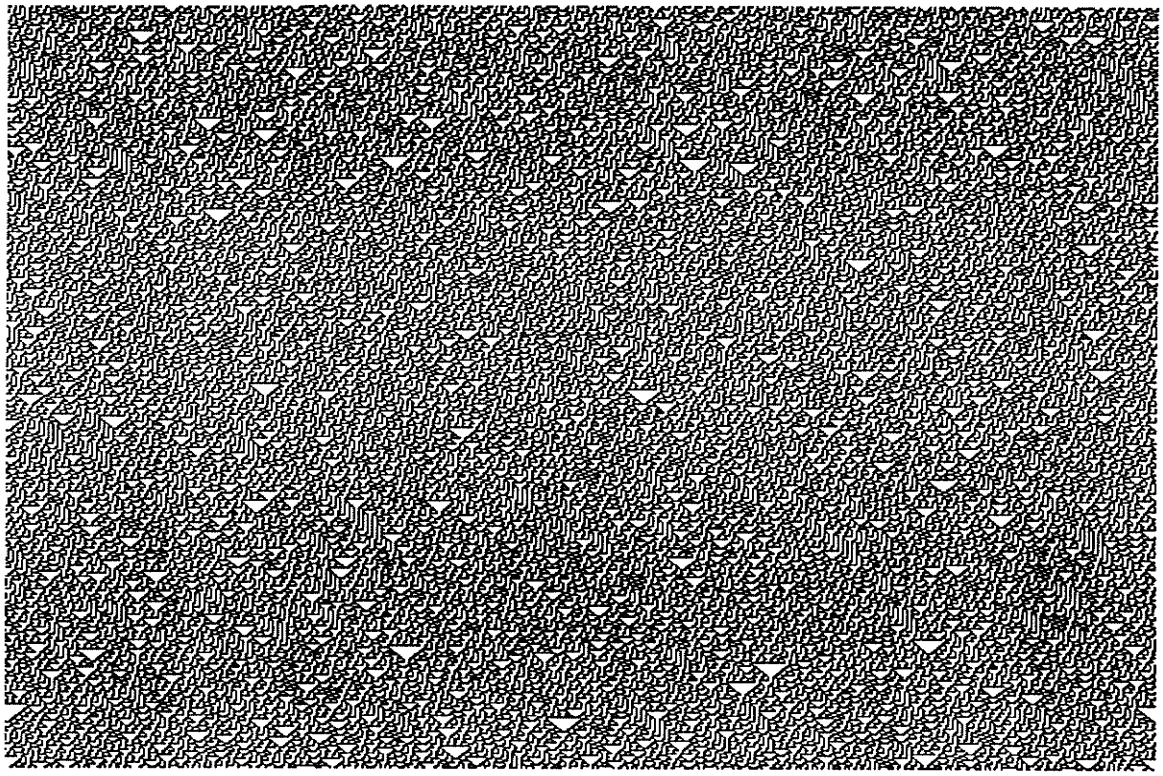


Figure 2.18 : *420 time steps in the state - time diagram of a 500 site rule 30 cellular automaton. Cyclic boundary conditions; random initial state.*

complicated transformations on the initial starting value. To follow the evolution of a particular cellular automaton one must use computations which are much more complicated than the rule of operation unless one is aware of the particular CA rule being used. Therefore, to predict the next state of the cellular automaton often takes more time than the cellular automaton required to evolve to it. In fact, Wolfram suspects that the evolution of many class 3 cellular automata are as computationally sophisticated as any (physically realisable) system and so, are computationally irreducible [Wolfram1985a], [Wolfram1984d]. Therefore, its outcome can be found only by observation or simulation. It is possible with some class 3 CA rules to use special features inherent to the rule to make analysis, and therefore predictions, possible. An example of such a CA rule is rule 60 which is a linear rule and so, can be described algebraically [Martin1984]. However, most class 3 CA rules are nonlinear and require algorithms much more complicated than their own rules of operation in order to be described.

A distinction can be drawn between *homoplectic* behaviour, in which pseudorandom sequences of states are produced when pseudorandom input or initial states are used, and *autoplectic* behaviour, in which pseudorandom behaviour arises even from simple initial conditions. It is the autoplectic CA rules which are the most intertesting since they provide an independence of starting state. The novel approach taken in this

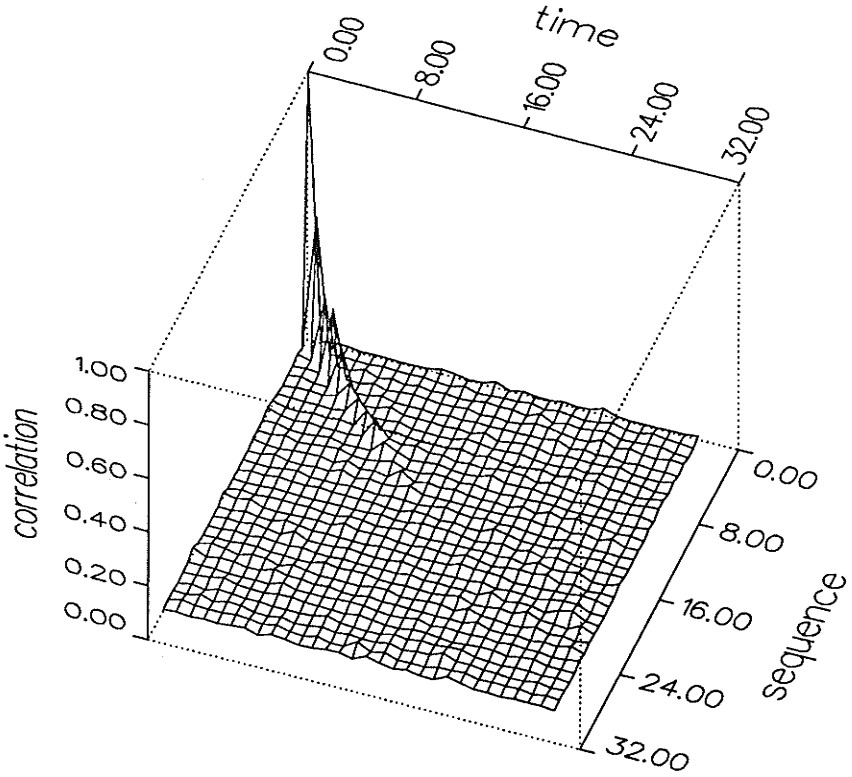


Figure 2.19 (a): *Cross-correlation of site values in a 30 site rule 30 cellular automaton.*

work towards parallel hardware pseudorandom number generation takes advantage of some of these recent discoveries by Wolfram. As before, for any of the following CA-based pseudorandom number generators to be considered pseudorandom we will require that the sequence of numbers generated pass the standard statistical random number tests given in Section 2.2.1.

2.4.2. CA rule 30

Consider a simple one-dimensional cellular automaton using rule 30; i.e.

$$a_i(t+1) = a_{i-1}(t) \oplus (a_i(t) \cup a_{i+1}(t)) \; . \qquad (2.30)$$

This particular CA rule has been investigated and is an example of a CA rule giving autoplectic behaviour in the sequence of site values, $a_i(t)$ [Wolfram1986a]. Therefore, each cell output can be considered as a pseudorandom bit sequence as may be observed in Fig. 2.16 where the output of a single cellular automaton site is given using the previously described raster scan technique. Figure 2.17 shows 420 time steps in the state - time diagram of a 500 site rule 30 cellular automaton with cyclic boundary conditions. Notice the triangular shapes which are randomly scattered throughout the state - time diagram. These triangular shapes are characteristic of many one-dimensional cellular automata. Also, it can be seen that the left edge of the cellular automaton exhibits a regular pattern which eventually dies out. This is an

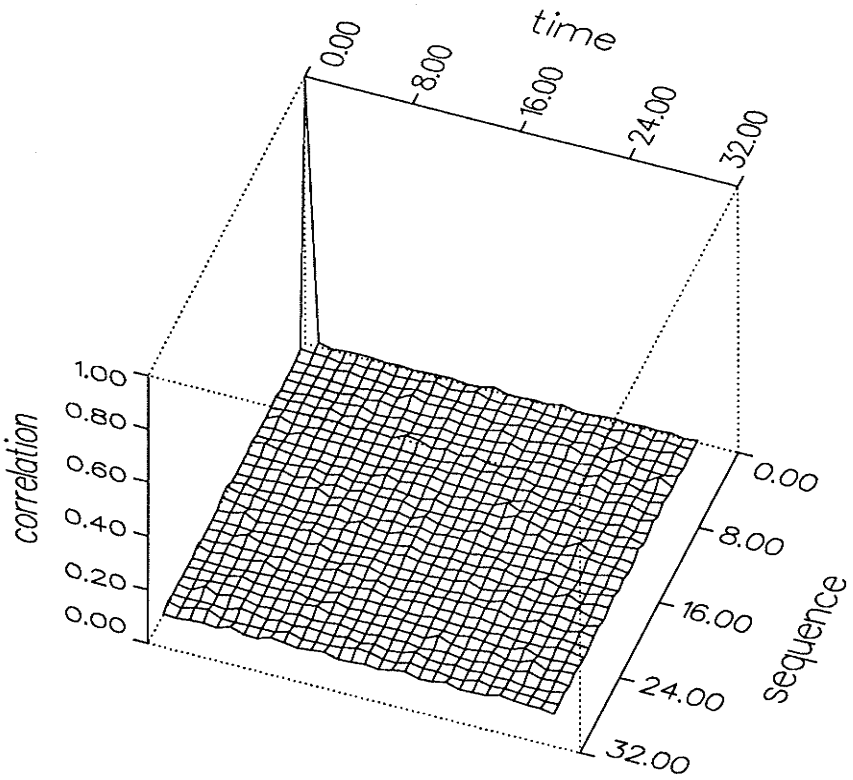


Figure 2.19 (b): *Cross-correlation of bit streams using a multiplicative congruential generator.*

example of the autoplectic behaviour of CA rule 30 (i.e. given a regular starting value as in Fig. 2.17 it eventually develops pseudorandom behaviour at each site). The right hand side of Fig. 2.17 evolves to randomness much faster than the left because of the asymmetry of this rule. In Fig. 2.18 the state - time diagram for CA rule 30 with a random initial state is shown. Notice that here the pseudorandom behaviour is retained over the entire cellular automaton showing an example of homoplectic behaviour.

The area used by a 30 bit rule 30 cellular automaton is $1.1 \times 10^6\ \mu m^2$ compared to $0.46 \times 10^6\ \mu m^2$ for a 30 bit LFSR. Therefore, a rule 30 cellular automaton yields an implementation which uses only 2.5 times the area of the parallel LFSR method (presently considered the most area efficient method but having unsatisfactory randomness). Additional advantages arise from the nearest neighbour communication properties of cellular automata. This avoids the global wiring of the LFSR (i.e. cellular automata can operate at higher speeds). Also, in the LFSR *exclusive–or* gates are used as adders in the feedback path in positions, or taps, that change as the LFSR length is modified. This leads to irregular circuit implementations, which do not occur in cellular automata since all sites are the same (i.e. if the size changes then sites can be added or removed with no other design changes).

It is rather obvious that if words are made by considering all sites, or bits, in parallel then the words do not constitute a truly pseudorandom sequence due to the

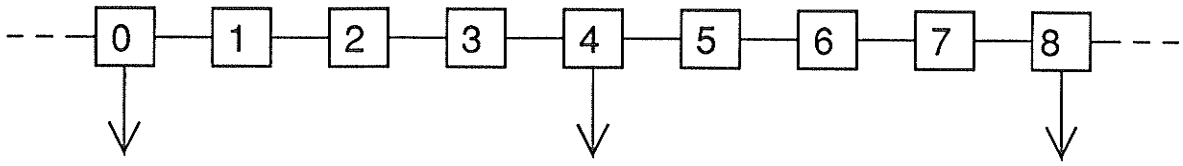


Figure 2.20 : *Definition of site spacing, here* $\gamma = 3$.

triangular features of Figs. 2.17 and 2.18. This is made more apparent by considering the auto and cross-correlation values for two pseudorandom number sequences of 30 bit words as shown in Fig. 2.19. In Fig. 2.19(a) the sequence is generated by CA rule 30 and shows that the correlation of values one site over on the next time step has a correlation of about 0.50 which implies that words formed from this register by considering every site in parallel are not independent. However, this correlation dies out and a bit stream 5 sites to the left or right is no longer correlated. For comparison the auto and cross-correlation for a sequence of 30 bit words generated by a multiplicative congruential generator is given in Fig. 2.19(b). Notice that there is no auto or cross-correlation between adjacent bit streams or time steps. Rule 30 compares very favourably with the auto and cross-correlation of the parallel LFSR generator shown in Fig. 2.11(bottom). For the parallel LFSR the correlation does not die out with site separation as in CA rule 30. This implies that any application in which the outputs of a LFSR are being used in parallel to produce pseudorandom numbers is better served by CA rule 30. In fact, this approach will be exploited in the CALBO (Cellular Automaton Logic Block Observer) built-in self-test circuit described in Chapter 4.

The fact that the correlation dies out over time with CA rule 30 also implies that if we use *site spacing* between output sites, as in Fig. 2.20, it would be possible to decorrelate adjacent bit streams in the output word. We will define a site spacing parameter, γ, where the value of γ will be the number of sites between outputs in the cellular automaton. For example, in Fig. 2.20 we have $\gamma = 3$. Therefore, as γ is increased we expect the cross-correlation between adjacent bit streams in the pseudorandom numbers to be reduced. In Fig. 2.21 the cross-correlation for various values of γ is shown. Note that for $\gamma \geq 4$ we have reduced the cross-correlation between adjacent bit streams to less than 10% (i.e. adjacent bit streams can now be considered uncorrelated). However, cross-correlation is not the only test for randomness. In Table 2.4 the random number test results for various values of γ are given. Notice that both the evenly weighted average and worst case failure metrics steadily decrease until a spacing of $\gamma = 3$. At this point the CA rule 30 based PRNG is performing as well as one of the standard algorithmic generators shown in Table 2.2. However, at $\gamma = 3$ the bit sequence correlation test is still consistently failed. Finally, for $\gamma = 4$ we see that this test is passed. Therefore, we will state that a value of $\gamma \geq 4$ is required to produce a *good* set of parallel pseudorandom sequences using CA rule 30. However,

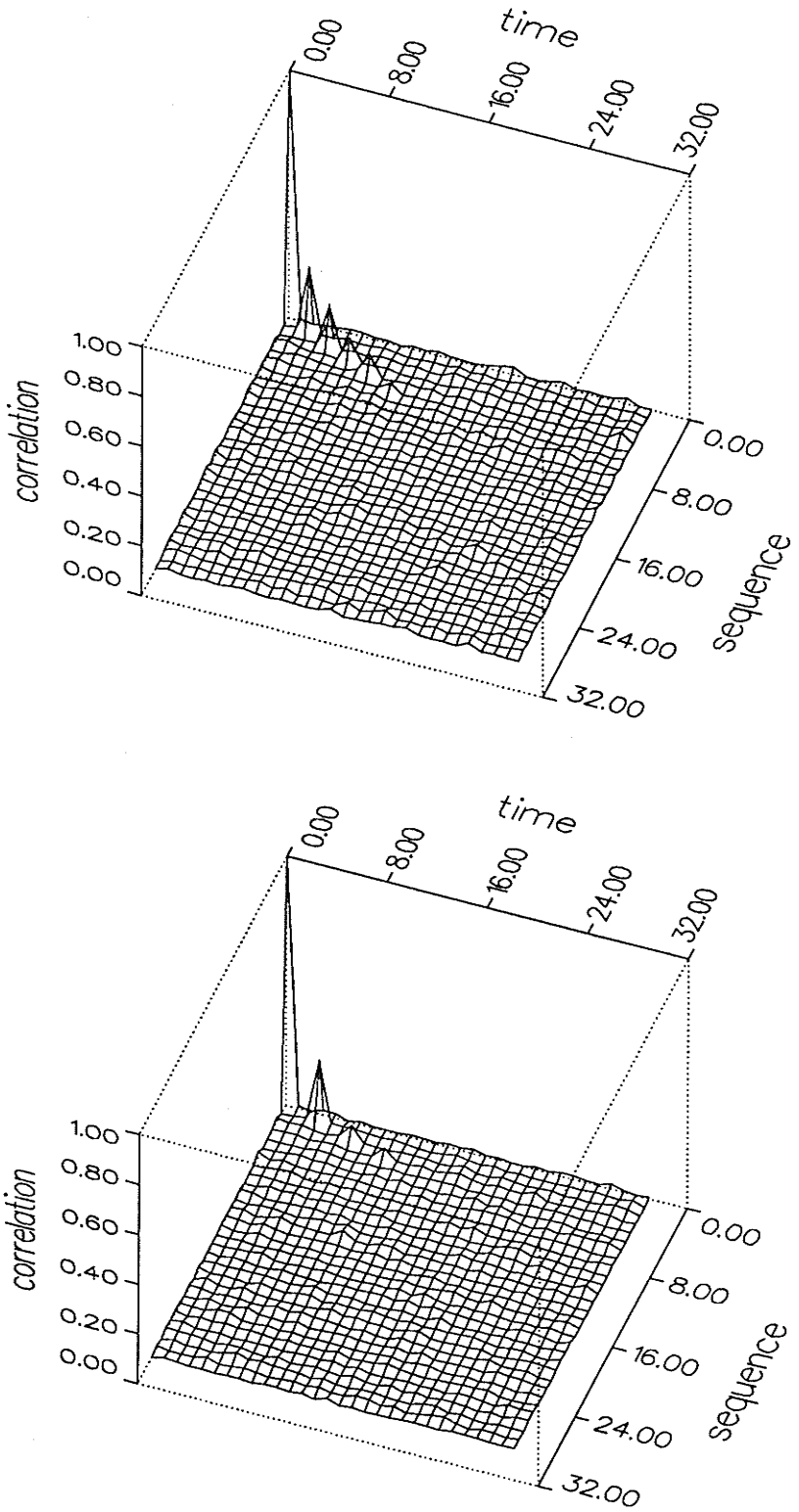


Figure 2.21 (a): *Cross-correlation between adjacent output bit streams in a 30 site rule 30 cellular automaton with site spacing, (top) $\gamma = 1$; (bottom) $\gamma = 2$.*

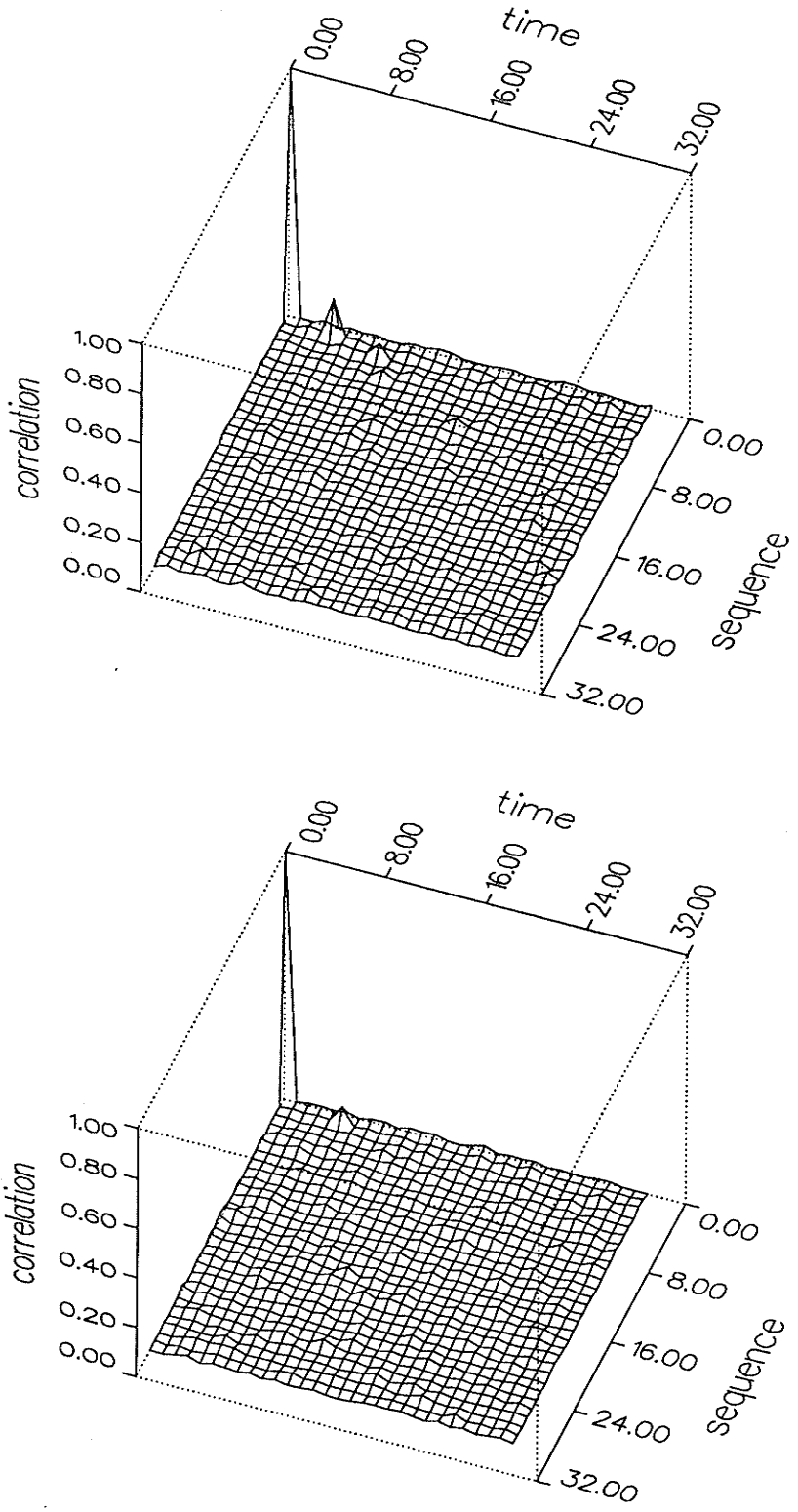


Figure 2.21 (b): *Cross-correlation between adjacent output bit streams in a 30 site rule 30 cellular automaton with site spacing, (top)* $\gamma = 3$*; (bottom)* $\gamma = 4$.

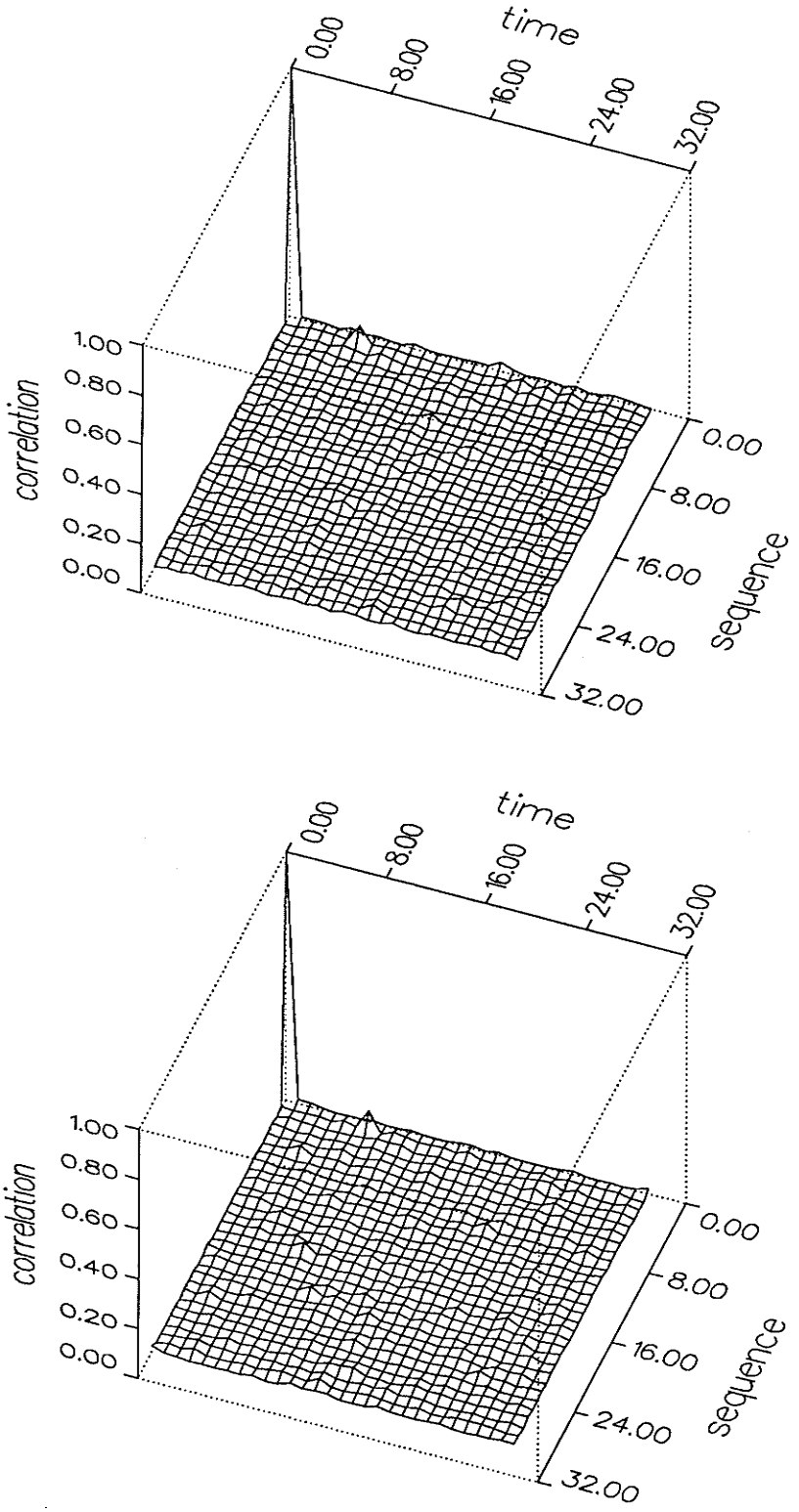


Figure 2.21 (c): *Cross-correlation between adjacent output bit streams in a 30 site rule 30 cellular automaton with site spacing, (top) $\gamma = 5$; (bottom) $\gamma = 6$.*

Sequence length = 1,000

Test	γ = 0		γ = 1		γ = 2		γ = 3		γ = 4		γ = 5		γ = 6	
mod	10	100	10	100	10	100	10	100	10	100	10	100	10	100
1	100	100	74	74	70	100	100	100	100	100	100	100	100	100
2	100	75	100	100	100	83	100	100	100	77	100	100	100	70
3	75	skip	70	skip	70	skip	100	skip	100	skip	78	skip	100	skip
4	100	skip	100	skip	100	skip	75	skip	75	skip	100	skip	100	skip
5	75	100	100	100	100	100	76	100	100	100	100	100	100	100
6	100	100	100	100	100	100	100	100	100	100	100	100	100	100
7	100	100	100	74	100	100	100	100	51	100	74	76	100	100
8	100	100	100	100	100	100	100	100	100	100	78	78	100	100
9	100	100	100	100	100	100	100	100	100	100	78	100	100	100
10	75	100	77	100	70	100	100	100	100	100	54	100	100	100
11	100	100	100	100	100	100	100	100	100	100	100	76	100	100
12	100	skip	79	skip	83	skip	100	skip	100	skip	100	skip	77	skip
13	100	100	100	100	100	100	100	100	100	100	74	100	100	100
14	100	100	100	100	100	77	100	78	100	100	100	100	77	100
15	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100
16	skip	100	skip	100	skip	100	skip	100	skip	100	skip	78	skip	100
17	skip	100	skip	44	skip	100	skip	100	skip	100	skip	100	skip	77
18	skip	100	skip	100	skip	100	skip	100	skip	100	skip	72	skip	100
19	skip	100	skip	100	skip	70	skip	100	skip	100	skip	100	skip	100
20	100	75	74	79	100	100	100	100	100	100	100	74	70	70
21	25	75	44	100	53	47	76	49	75	74	76	52	30	76
22	0.51		0.28		0.25		0.16		0.08		0.07		0.13	
23	Fail		Fail		Fail		Pass		Pass		Pass		Pass	
24	5	4	5	5	4	4	3	3	2	1	4	3	3	2
25	3.50	2.75	3.82	3.29	3.54	3.23	1.73	1.73	0.99	0.49	1.88	1.94	1.46	1.07

Sequence length = 10,000

Test	γ = 0		γ = 1		γ = 2		γ = 3		γ = 4		γ = 5		γ = 6	
mod	10	100	10	100	10	100	10	100	10	100	10	100	10	100
1	66	66	77	77	76	100	100	100	100	77	100	100	100	100
2	66	0	100	100	79	100	100	100	49	100	100	71	100	100
3	61	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100	skip
4	100	skip	100	skip	100	skip	100	skip	100	skip	100	skip	76	skip
5	100	100	100	100	100	100	100	100	100	100	100	77	100	72
6	100	100	100	100	79	76	100	76	100	100	100	77	100	100
7	100	100	100	77	100	100	100	100	77	100	100	100	100	76
8	73	100	100	100	76	100	100	100	100	74	76	100	100	76
9	100	100	68	100	100	100	100	73	100	100	71	71	100	100
10	66	66	100	51	100	100	80	100	100	100	100	100	100	100
11	100	80	68	68	69	100	100	100	100	100	100	100	100	100
12	80	skip	100	skip	100	skip	73	skip	100	skip	100	skip	52	skip
13	100	100	100	100	100	100	100	100	77	100	76	100	100	76
14	100	73	68	77	100	79	100	100	100	100	76	100	100	72
15	skip	66	skip	83	skip	100	skip	56	skip	100	skip	100	skip	76
16	skip	100	skip	72	skip	76	skip	100	skip	100	skip	100	skip	100
17	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100
18	skip	100	skip	100	skip	100	skip	100	skip	77	skip	71	skip	100
19	skip	100	skip	100	skip	79	skip	73	skip	73	skip	100	skip	100
20	80	53	77	100	100	100	100	73	100	77	100	100	100	100
21	0	46	100	68	100	69	80	100	27	77	100	71	100	48
22	0.50		0.26		0.25		0.16		0.07		0.07		0.05	
23	Fail		Fail		Fail		Pass		Pass		Pass		Pass	
24	6	7	5	5	4	4	3	4	3	4	2	4	1	3
25	5.08	5.50	3.42	4.27	3.21	3.21	1.67	2.49	1.70	1.45	1.01	1.62	0.72	2.04

Table 2.4: *Random number test results for CA rule 30 with various site spacing values, γ.*

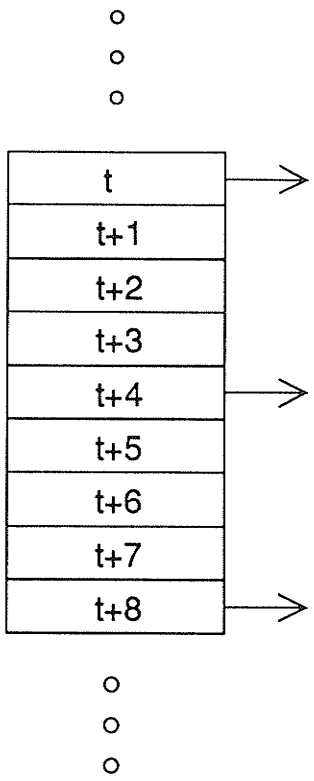


Figure 2.22 : *Definition of time spacing, here* $\beta = 3$.

as we will later see, the performance of CA rule 30 based PRNGs for lesser values of γ is quite good and is suitable in some applications.

Another method of removing the cross-correlation between adjacent cellular automaton sites is to use *time spacing*. Here the cellular automaton is clocked several times between each pseudorandom number used, as shown in Fig. 2.22. As before, consider a time spacing parameter, β, where the value of β is the time step spacing between output numbers. For example, in Fig. 2.22 we have $\beta = 3$. The cross-correlation for various values of β, as shown in Fig. 2.23, indicates that adjacent bit streams become uncorrelated (i.e. less than 10%) for $\beta \geq 4$. In Table 2.5 the results of the random number tests for various values of β are given. As in the site spacing case, the evenly weighted average and worst case failure metrics both steadily decline as β is increased. We will consider $\beta \geq 4$ to be required for good pseudorandom sequences since for smaller values of β the bit sequence correlation test is consistently failed. However, as in the site spacing situation, lesser values of β yield CA rule 30 based PRNGs which still deliver good performance and may be useful for some applications.

To compare the implementations in silicon of the two methods (site spacing and time spacing) using CA rule 30 to generate truly pseudorandom sequences, we will again consider the AT metric. The area required by a rule 30 cellular automaton with $\gamma = 0$ is $1.1 \times 10^6\ \mu m^2$. If we consider the minimum values of γ and β required to produce a satisfactorily pseudorandom sequence (i.e. $\gamma = 4$ and $\beta = 4$) we see that the

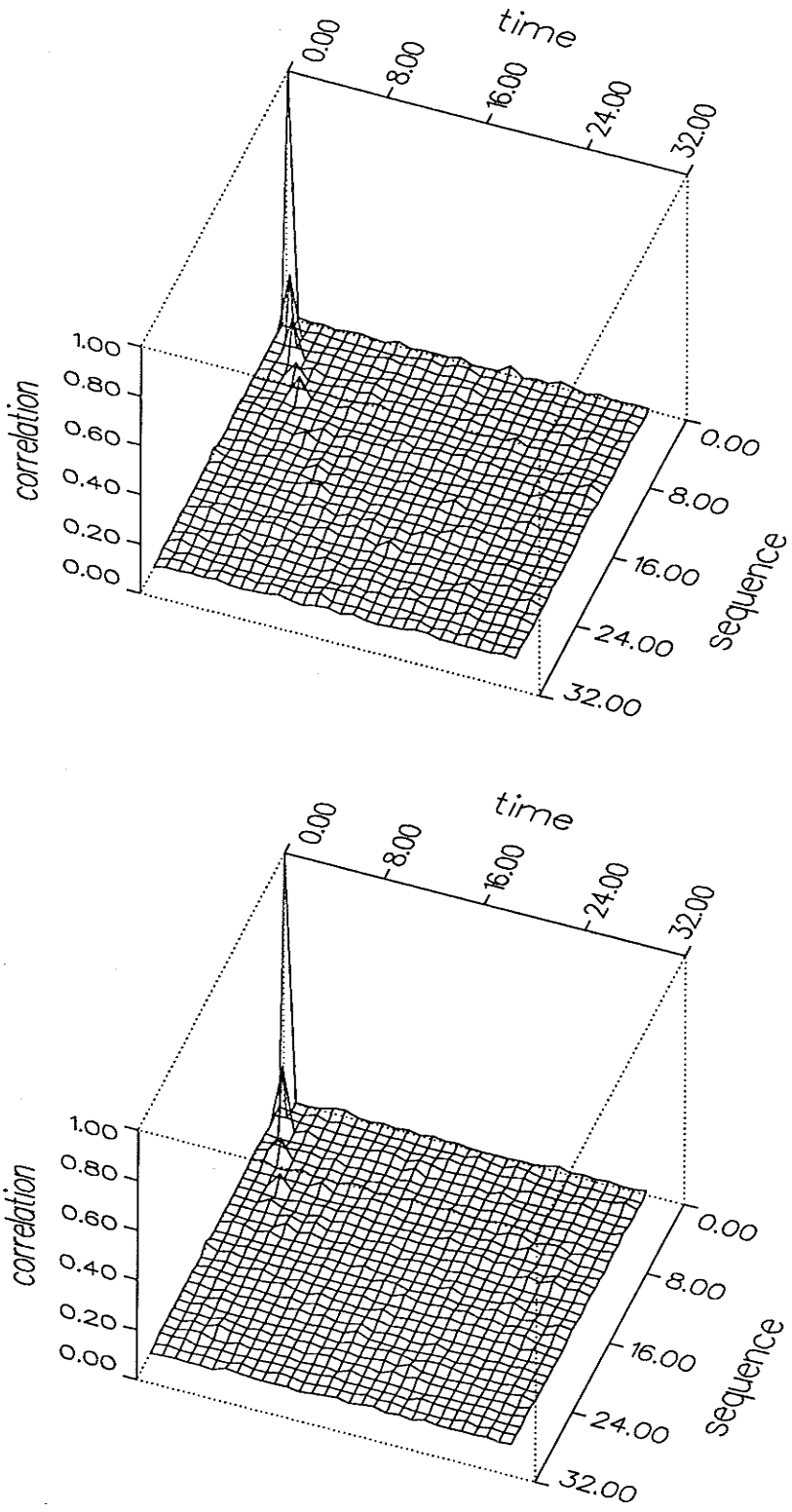


Figure 2.23 (a): *Cross-correlation between adjacent output bit streams in a 30 site rule 30 cellular automaton with time spacing, (top)* $\beta = 1$*; (bottom)* $\beta = 2$.

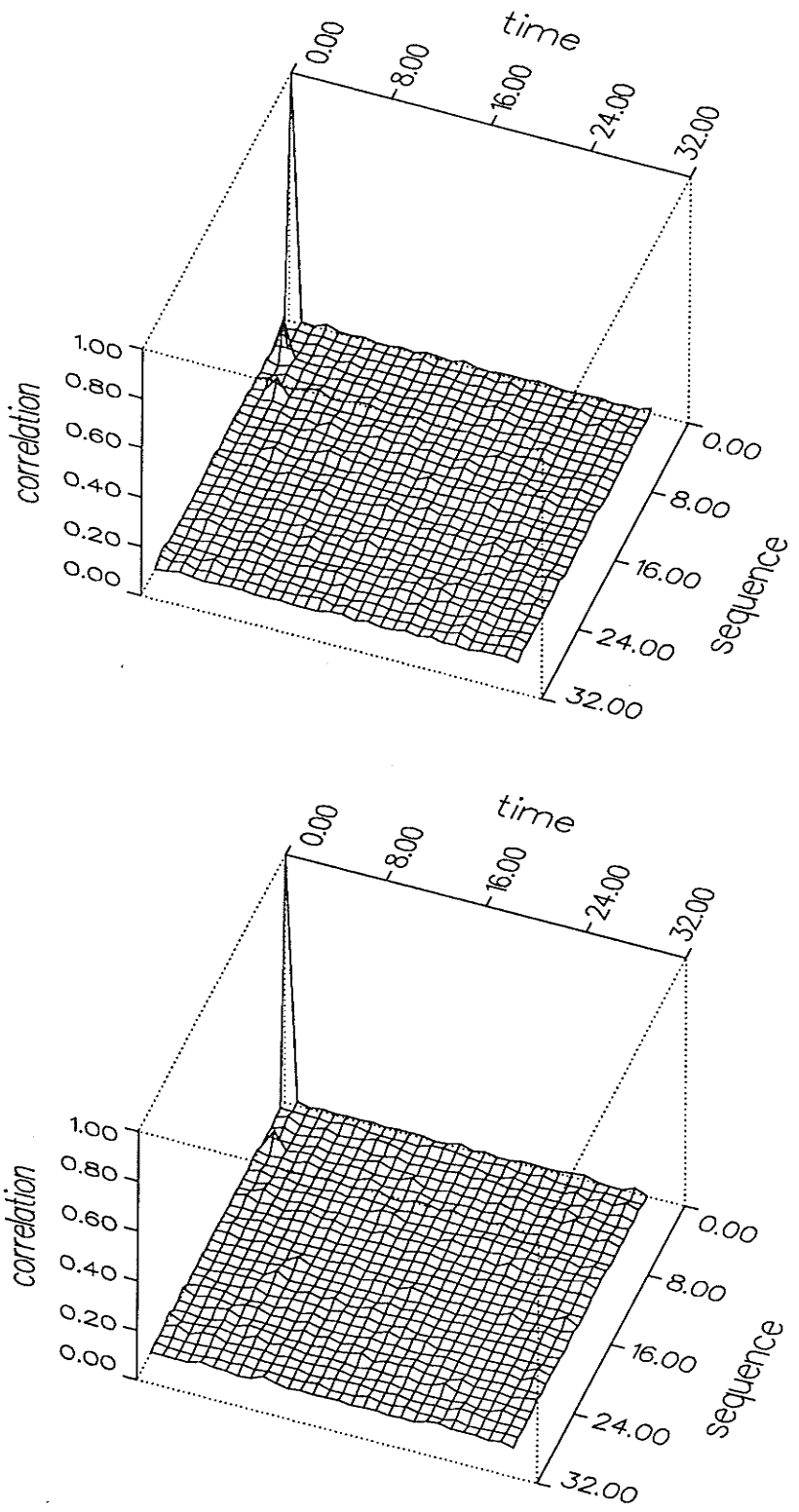


Figure 2.23 (b): *Cross-correlation between adjacent output bit streams in a 30 site rule 30 cellular automaton with time spacing, (top)* $\beta = 3$*; (bottom)* $\beta = 4$.

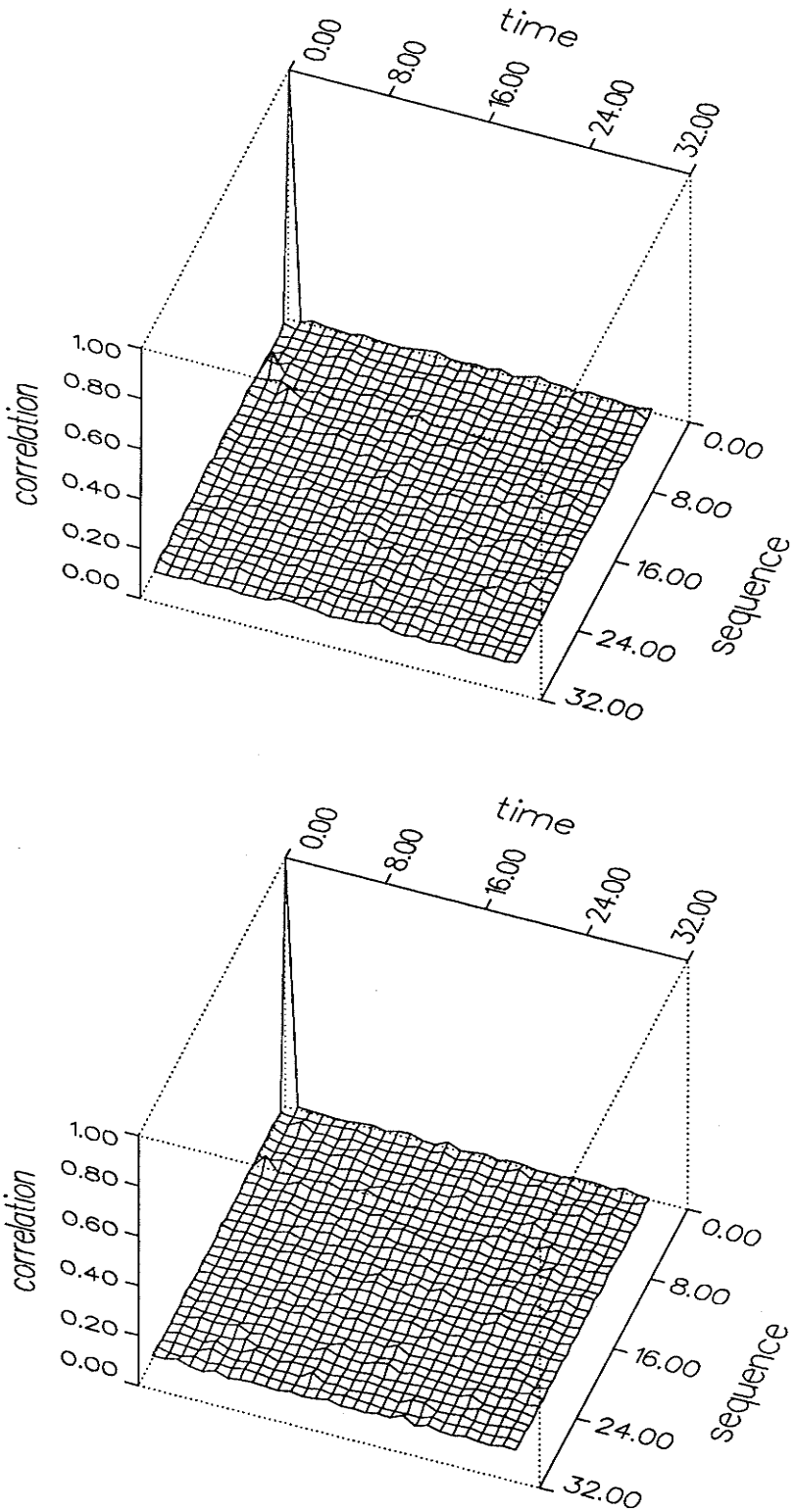


Figure 2.23 (c): *Cross-correlation between adjacent output bit streams in a 30 site rule 30 cellular automaton with time spacing, (top)* $\beta = 5$*; (bottom)* $\beta = 6$.

Sequence length = 1,000

Test	β = 0		β = 1		β = 2		β = 3		β = 4		β = 5		β = 6	
mod	10	100	10	100	10	100	10	100	10	100	10	100	10	100
1	100	100	100	100	100	61	100	100	100	83	100	100	84	100
2	100	85	87	100	100	100	74	100	100	100	69	100	100	65
3	81	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100	skip
4	100	skip	78	skip	100	skip	100	skip	100	skip	83	skip	79	skip
5	85	100	100	100	100	100	100	74	100	100	100	100	72	65
6	100	100	100	100	100	100	100	100	83	83	69	100	100	100
7	100	100	78	100	100	100	100	100	100	56	100	100	100	100
8	100	100	100	64	76	100	100	100	100	100	100	100	100	100
9	100	100	100	100	100	100	100	100	100	100	100	100	100	100
10	85	100	100	100	100	100	100	100	100	100	100	100	100	100
11	100	100	78	100	100	100	100	100	48	100	100	100	100	100
12	100	skip	100	skip	100	skip	100	skip	100	skip	69	skip	100	skip
13	100	100	100	100	100	100	100	100	100	100	69	100	100	100
14	100	100	87	100	100	100	100	100	100	38	100	100	100	65
15	skip	100	skip	71	skip	100	skip	100	skip	79	skip	69	skip	100
16	skip	100	skip	100	skip	100	skip	50	skip	100	skip	100	skip	65
17	skip	100	skip	100	skip	76	skip	100	skip	100	skip	100	skip	100
18	skip	100	skip	71	skip	100	skip	66	skip	100	skip	83	skip	79
19	skip	100	skip	71	skip	100	skip	100	skip	100	skip	100	skip	65
20	100	85	100	100	100	37	100	100	100	100	100	100	37	100
21	19	81	78	13	61	100	40	74	100	52	67	50	44	28
22	0.52		0.24		0.25		0.15		0.11		0.13		0.07	
23	Fail		Fail		Fail		Pass		Pass		Pass		Pass	
24	5	4	6	6	3	4	3	4	2	3	4	2	2	6
25	3.30	2.49	3.14	4.10	2.63	3.26	1.86	2.36	0.69	2.09	1.74	0.98	1.84	2.68

Sequence length = 10,000

Test	β = 0		β = 1		β = 2		β = 3		β = 4		β = 5		β = 6	
mod	10	100	10	100	10	100	10	100	10	100	10	100	10	100
1	75	75	100	70	100	79	100	100	67	67	81	100	100	100
2	75	0	100	100	100	79	100	78	100	77	81	100	80	100
3	55	skip	76	skip	100	skip	100	skip	100	skip	81	skip	100	skip
4	100	skip	100	skip	100	skip	100	skip	77	skip	53	skip	100	skip
5	100	100	54	100	100	100	100	100	100	100	100	100	70	100
6	100	100	100	100	100	100	100	100	100	56	100	100	100	70
7	100	100	100	100	100	100	100	100	100	100	72	100	100	100
8	70	100	100	100	70	100	100	100	100	100	100	81	74	100
9	100	100	100	100	100	100	100	100	100	100	100	100	100	100
10	75	75	100	100	100	100	78	100	80	100	100	100	100	100
11	100	78	100	100	100	100	100	74	100	100	72	100	100	100
12	78	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100	skip
13	100	100	100	100	100	100	100	70	100	100	100	100	80	100
14	100	70	70	100	78	100	100	100	100	100	100	100	100	100
15	skip	75	skip	76	skip	100	skip	78	skip	100	skip	100	skip	100
16	skip	100	skip	100	skip	100	skip	100	skip	100	skip	71	skip	100
17	skip	100	skip	100	skip	78	skip	100	skip	67	skip	72	skip	100
18	skip	100	skip	100	skip	100	skip	100	skip	100	skip	81	skip	100
19	skip	100	skip	100	skip	70	skip	100	skip	100	skip	100	skip	100
20	78	48	100	100	100	79	78	100	67	100	100	100	80	100
21	0	53	24	44	100	70	100	70	100	100	72	76	56	44
22	0.51		0.24		0.25		0.16		0.07		0.08		0.07	
23	Fail		Fail		Fail		Pass		Pass		Pass		Pass	
24	6	7	4	4	3	5	3	3	2	2	4	2	4	2
25	4.94	5.26	3.76	3.10	2.52	3.45	1.44	2.30	1.09	1.33	1.88	1.19	1.60	0.86

Table 2.5: *Random number test results for CA rule 30 with various time spacing values, β.*

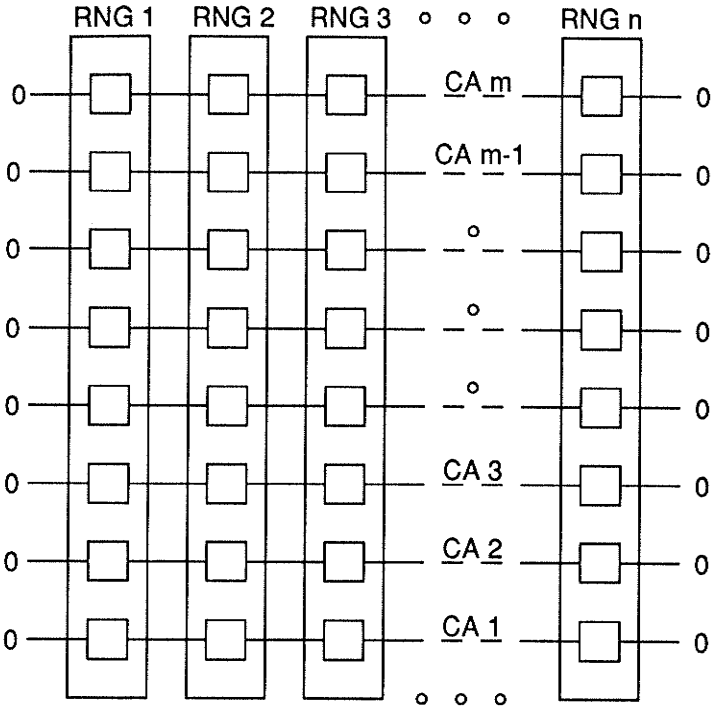


Figure 2.24 (a): *A parallel pseudorandom sequence generator using m rule 30 cellular automata with null boundary conditions.*

two values are the same. Therefore, the use of site spacing to produce a good pseudorandom sequence will require 5 times more area than that with time spacing, or $5.6 \times 10^6\ \mu m^2$. However, time spacing will increase the time to generate a pseudorandom number by the same factor. In fine-grained parallel processing networks the choice of time spacing versus site spacing is application dependent, and will depend on the relative sizes of the other computing hardware at each processor. For example, if each processor employs very simple logic, the size of the PRNG may be the dominant consideration. On the other hand, if the processors contain more complicated circuitry than the pseudorandom generator, an approach based on site spacing may be more suitable. The site spacing generator may also be preferred even with simple processor sites, as these processors generally can make use of a new pseudorandom number on each clock cycle. More complicated processors usually require several clock cycles to process each new pseudorandom number, in which case the extra time required for the time spacing generator would provide no penalty. In the remaining discussions, only the site spacing method will be further considered. The time spaced approach may be substituted provided the appropriate adjustments are made.

We proceed to overcome the cross-correlation problems in CA rule 30 by using site spacing. The site spacing can automatically be obtained in the parallel processor

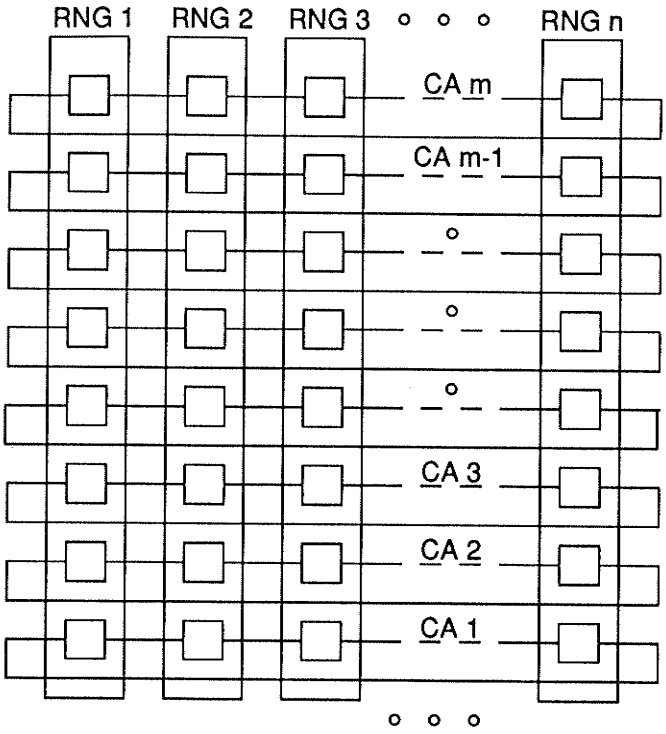


Figure 2.24 (b): *A parallel pseudorandom sequence generator using m rule 30 cellular automata with cyclic boundary conditions.*

architectures of Fig. 2.24. Here we consider each m bit pseudorandom word at processor i to consist of bits $a_i^0(t)$, $a_i^1(t)$, $\cdots$, $a_i^{m-1}(t)$, where the a_i^j are not taken from adjacent sites. At the end of the cellular automaton at level k we have a choice of cyclic or null boundary conditions, or better still, we may allow the end of cellular automaton k to join the beginning site of cellular automaton $k + 1$. The establishment of null boundary conditions, as in Fig. 2.24(a), prevents global wiring but from Table 2.6 we see that cyclic boundary conditions provide much longer cycle lengths for the same size register. In fact, when null boundary conditions are used, the cycle lengths are very short. However, the use of paths to the cycles can provide a sufficiently long sequence of pseudorandom numbers to make null boundary conditions a feasible layout. A more complete discussion comparing null and cyclic boundary conditions follows later in the chapter.

The need for cyclic boundary conditions as in Fig. 2.24(b) is not a major consideration since the register can be made to wind around the processors in such a way that the two ends of the cellular automaton become adjacent. Using cyclic boundary conditions, Table 2.6 indicates that a cellular automaton of at least 51 sites is needed to produce sequences with cycle lengths of more than 1×10^9. Finally, if the end of cellular automaton k is joined to the beginning of cellular automaton $k + 1$, as in Fig. 2.25, we will further lengthen the cycle lengths for the sequence. The only

	CA Rule 30 cyclic		CA Rule 30 null		Hybrid CA and LFSR	
N	C_N	$\log_2 C_N$	C_N	$\log_2 C_N$	C_N	$\log_2 C_N$
4	8	3.0	2	1.0	15	3.9
5	5	2.3	1	0.0	31	5.0
6	1	0.0	2	1.0	63	6.0
7	63	6.0	1	0.0	127	7.0
8	40	5.3	2	1.0	255	8.0
9	171	7.4	1	0.0	511	9.0
10	15	3.9	2	1.0	1,023	10.0
11	154	7.3	1	0.0	2,047	11.0
12	102	6.7	2	1.0	4,095	12.0
13	832	9.7	1	0.0	8,191	13.0
14	1,428	10.5	2	1.0	16,383	14.0
15	1,455	10.5	1	0.0	32,767	15.0
16	6,016	12.6	2	1.0	64,535	16.0
17	10,845	13.4	1	0.0	131,071	17.0
18	2,844	11.5	2	1.0	262,143	18.0
19	3,705	11.9	1	0.0	524,827	19.0
20	6,150	12.6	2	1.0	1,048,575	20.0
21	2,793	11.4	1	0.0	2,097,151	21.0
22	3,256	11.7	2	1.0	4,194,303	22.0
23	38,249	15.2	1	0.0	8,388,607	23.0
24	184,040	17.5	2	1.0	16,777,213	24.0
25	588,425	19.2	1	0.0	33,554,431	25.0
26	312,156	18.3	2	1.0	67,108,865	26.0
27	67,554	16.0	1	0.0	134,217,727	27.0
28	249,165	17.9	2	1.0	268,435,455	28.0
29	1,466,066	20.5	1	0.0	536,870,911	29.0
30	306,120	18.2	2	1.0	1,073,741,823	30.0
31	2,841,150	21.4	1	0.0	2,147,483,647	31.0
32	2,002,272	20.9	2	1.0	4,294,967,295	32.0
33	2,038,476	21.0	1	0.0	8,589,934,591	33.0
34	5,656,002	22.4	2	1.0	17,179,869,183	34.0
35	18,480,630	24.1	1	0.0	34,359,738,367	35.0
36	2,237,472	21.1	2	1.0	68,719,476,735	36.0
37	49,276,415	25.6	1	0.0	$2^{37}-1$	37.0
38	9,329,228	23.2	2	1.0	$2^{38}-1$	38.0
39	961,272	19.9	1	0.0	$2^{39}-1$	39.0
40	19,211,080	24.2	2	1.0	$2^{40}-1$	40.0
41	51,151,354	25.6	1	0.0	$2^{41}-1$	40.0
42	109,603,410	26.7	2	1.0	$2^{42}-1$	42.0
43	93,537,212	26.5	1	0.0	$2^{43}-1$	43.0
44	192,218,312	27.5	2	1.0	$2^{44}-1$	44.0
45	75,864,495	26.2	1	0.0	$2^{45}-1$	45.0
46	261,598,274	28.0	2	1.0	$2^{46}-1$	46.0
47	811,284,813	29.6	1	0.0	$2^{47}-1$	47.0
48	3,035,918,676	31.5	2	1.0	$2^{48}-1$	48.0
49	9,937,383,652	33.2	1	0.0	$2^{49}-1$	49.0
50	593,487,780	29.1	2	1.0	$2^{50}-1$	50.0
51	3,625,711,023	31.8	1	0.0	$2^{51}-1$	51.0
52	20,653,434,880	34.3	2	1.0	$2^{52}-1$	52.0
53	40,114,679,273	35.3	1	0.0	$2^{53}-1$	53.0
54	7,551,779,562	32.8	2	1.0	$2^{54}-1$	54.0

Table 2.6: *Maximum length cycles for cellular automata of length N under various various conditions (portions of this table are taken from [Wolfram1986a]). Here C_N represents the length of the maximum length cycle for a cellular automaton of length N.*

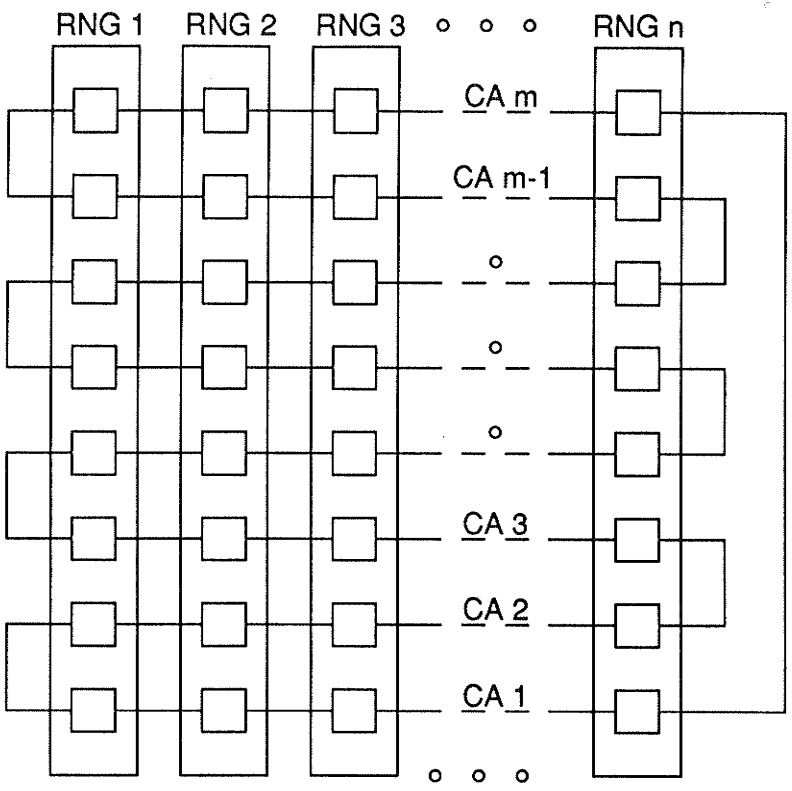


Figure 2.25 : *A parallel pseudorandom sequence generator using one rule 30 cellular automaton for all processors.*

caution to be observed is the cross-correlation of bit k between processors i, $i-1$, $i-2$, $i-3$, and $i-4$. However, if a spacing of $\gamma = 4$ is used, we can decorrelate bit k between processors. This also serves to further increase the cellular automaton length. For example, whereas the architectures of Fig. 2.24 will need at least 51 processors to generate a sequence of length $> 1 \times 10^9$, if $\gamma = 4$ is used only 11 processors will be required for the same cycle length.

Another method of generating pseudorandom numbers is to use a cellular automaton local to each processor. Here appropriate spacing should be used to ensure that the bits of the pseudorandom word are uncorrelated. Previously it was indicated that a 51 site cyclic rule 30 cellular automaton should be considered as a minimum in creating long sequences of pseudorandom numbers. One method of increasing the cycle lengths, if the PRNG must be local to each processor, is to connect each cellular automaton in the spirit of Fig. 2.25, where the end of one cellular automaton is attached to the start of another. As in the previous case, this will cause cross-correlation across bits between the processors. However, since site spacing is already present in each local cellular automaton, it should be possible to include similar spacing between the processors.

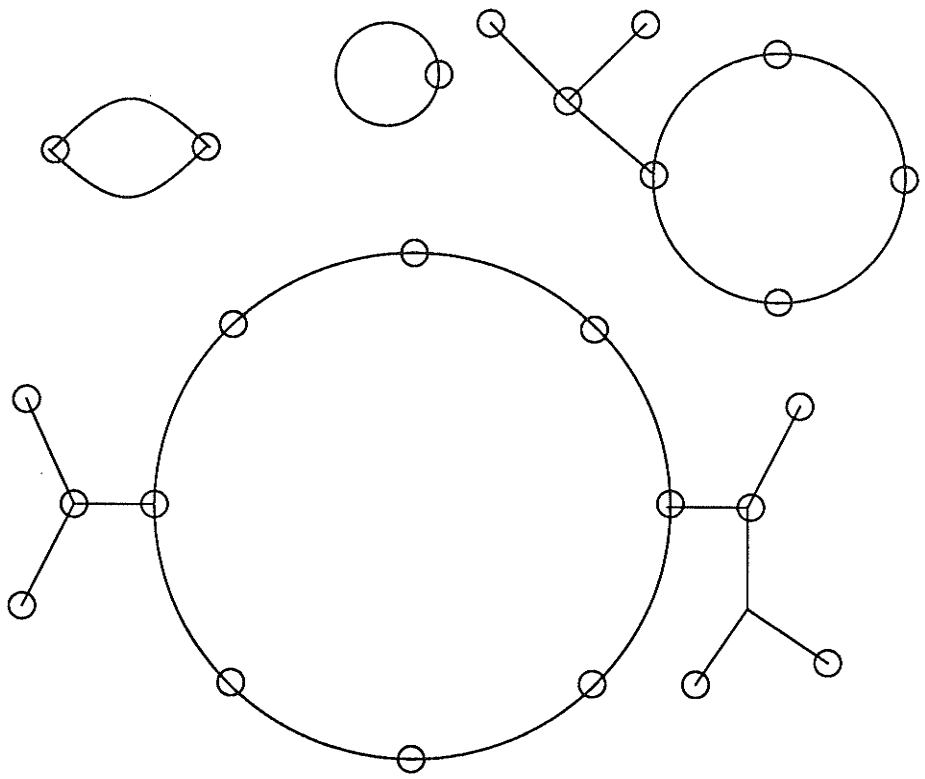


Figure 2.26 : *Typical cycles and paths to the cycles for CA rule 30 with cyclic boundary conditions.*

An important consideration in the use of any PRNG is the length of the sequence produced (i.e. after how many numbers does the sequence repeat). In most of the generators considered here the next value, X_{n+1}, depends solely on the previous sequence value, X_n. That is, once a value appears twice, the sequence begins to repeat. Note that usually we only consider a portion of each number in the sequence being produced (for example, we might use only the modulo d values or only certain bits of the sequence). These values may repeat without the complete number doing so. For register type generators, such as the LFSR and CA rule 30, this is especially true. The LFSR generators have a sequence length of $2^n - 1$, where n = length of the LFSR. However, CA rule 30 does not provide nearly as long a sequence. The sequences produced by CA rule 30 usually consist of cycles and paths to the cycles such as those shown in Fig. 2.26. The size of the cycles varies greatly in that some are large and others quite small. Table 2.6 shows the maximum cycle lengths of sequences resulting from various register sizes. For cyclic boundary conditions the cycle length increases at a rate approximately exponential with n. A least squares fit to the data shows that cycle length C_N can be approximated by [Wolfram1986a]

$$\log_2 C_N = 0.61(N + 1) \tag{2.31}$$

or

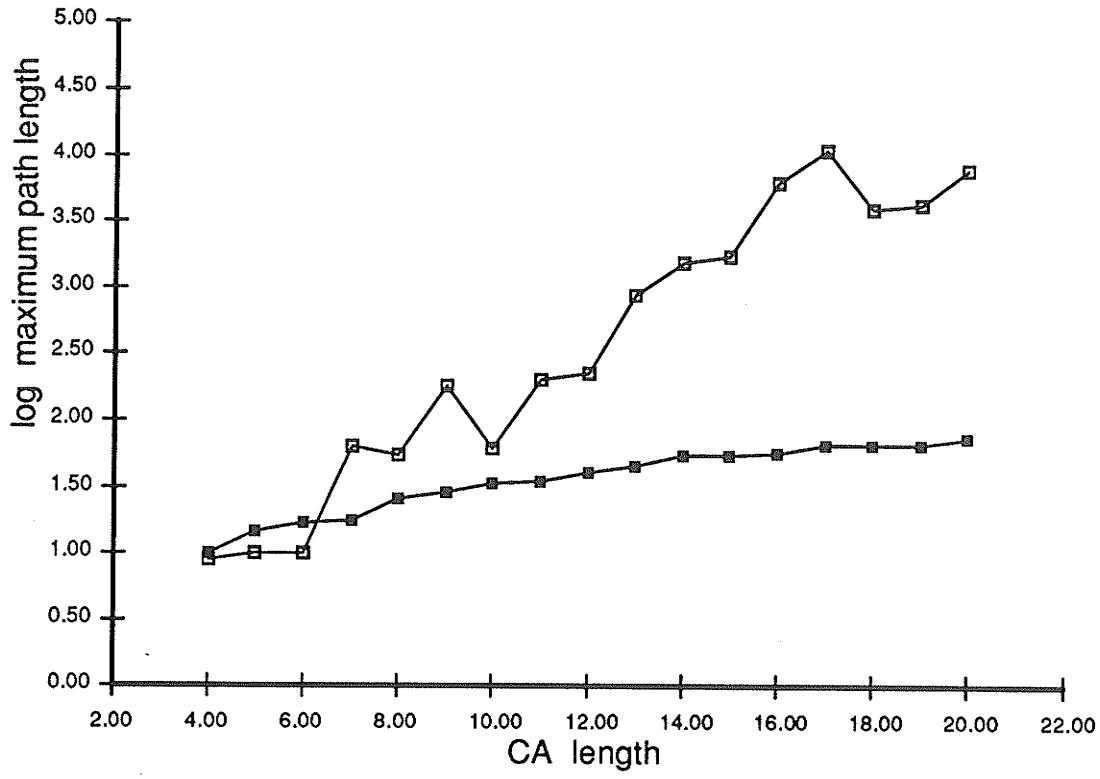


Figure 2.27 : *Maximum path length versus CA rule 30 size for both null (filled squares) and cyclic (empty squares) boundary conditions.*

$$C_N = 2^{0.61(N+1)} \quad . \tag{2.32}$$

This would initially appear to be quite poor as compared to the cycle length of $2^N - 1$ in an N bit LFSR. Yet, if a truly random mapping were used between the 2^N possible states in an N bit cellular automaton then as $N \rightarrow \infty$ the longest cycle is expected to have a length of $2^{\lambda N}$, where $\lambda = 0.62432 \cdots$ [Golomb1982], [Purdom1968] and the average length of the cycles would be $2^{N/2}$ [Harris1960]. It has been proposed that the exponent in Eqn. 2.32 may be related to the entropy of the sequence produced but this requires further investigation [Wolfram1986a]. CA rule 30 with null boundary conditions has very small cycles in that even sizes have a maximum cycle length of two and odd sizes have a maximum cycle length of one. Another peculiar property of this case is that there is only one maximal length cycle (i.e. all paths lead to the same short cycle) and the zero cycle is not entered by any path.

The lengths of the various paths in CA rule 30 with null boundary conditions differ widely. Therefore, if a path is sufficiently long, it may be possible to begin near the extremes in the path tree, and still provide a long sequence of pseudorandom numbers. In Fig. 2.27 a plot of maximum possible path length versus cellular automaton size is shown. Notice that the path length for cyclic boundary conditions increases rapidly mainly due to the increase in maximum cycle length. However, for null

	Cyclic			Null		
N	P_N	$\log_2 P_N$	start	P_N	$\log_2 P_N$	start
4	9	3.17	0011	10	3.32	0001
5	10	3.32	00011	15	3.91	00010
6	10	3.32	000011	17	4.09	000001
7	65	6.02	0000011	18	4.17	0100001
8	56	5.81	00111011	26	4.70	01011110
9	184	7.52	000111011	29	4.86	000100110
10	61	5.93	0001110011	34	5.09	0001001111
11	209	7.71	00111110111	35	5.13	00000000001
12	228	7.83	000011100011	41	5.36	000100110011
13	877	9.78	0000110111111	45	5.49	0001001110011
14	1555	10.60	00000110000111	54	5.75	00000000000001
15	1776	10.79	001101110111111	56	5.81	000000000001010
16	6269	12.61	0011011101110111	58	5.86	0001001100001100
17	11208	13.45	00001101111100011	67	6.07	00000000010111110
18	3981	11.96	000001110001100111	67	6.07	000100110011111001
19	4358	12.09	0000001110111111011	67	6.07	0101101111100001111
20	7886	12.95	00001100001101110011	75	6.23	00000000010111011011

Table 2.7: *Maximum length and starting value of nonrepeating sequences for CA rule 30 with N sites for both cyclic and null boundary conditions. Starting points for cyclic boundary conditions were found by exhaustive simulation while those for null boundary conditions were found using a method due to Pries [Pries1988].*

N	Cycles	Frac. longest
4	1x8, 3x1	0.75
5	1x5, 1x1	0.94
6	3x1	1.00
7	1x63, 7x4, 1x1	0.60
8	1x40, 1x8, 3x1	0.88
9	1x171, 1x72, 1x1	0.81
10	2x15, 1x5, 3x1	0.82
11	1x154, 11x17, 1x1	0.76
12	4x102, 1x8, 4x3, 3x1	0.93
13	1x832, 1x260, 1x247, 1x91, 1x1	0.32
14	1x1428, 2x133, 1x112, 2x84, 1x63, 1x14, 3x1	0.84
15	1x1455, 5x30, 5x9, 15x7, 4x5, 1x1	0.93
16	1x6016, 1x4144, 3x40, 1x8, 3x1	0.50
17	1x10846, 1x1632, 1x867, 1x306, 1x136, 1x17, 1x1	0.96
18	1x2844, 6x186, 1x171, 1x72, 6x24, 3x1	0.82
19	1x3705, 1x247, 1x133, 1x38, 1x1	0.72
20	1x6756, 1x6691, 2x6150, 4x3420, 4x1715, 1x580, 5x68, 4x30, 2x15, 1x8, 1x5, 3x1	0.01

Table 2.8: *Cycles lengths for various size rule 30 cellular automata and the fraction of all states leading to the longest cycle.*

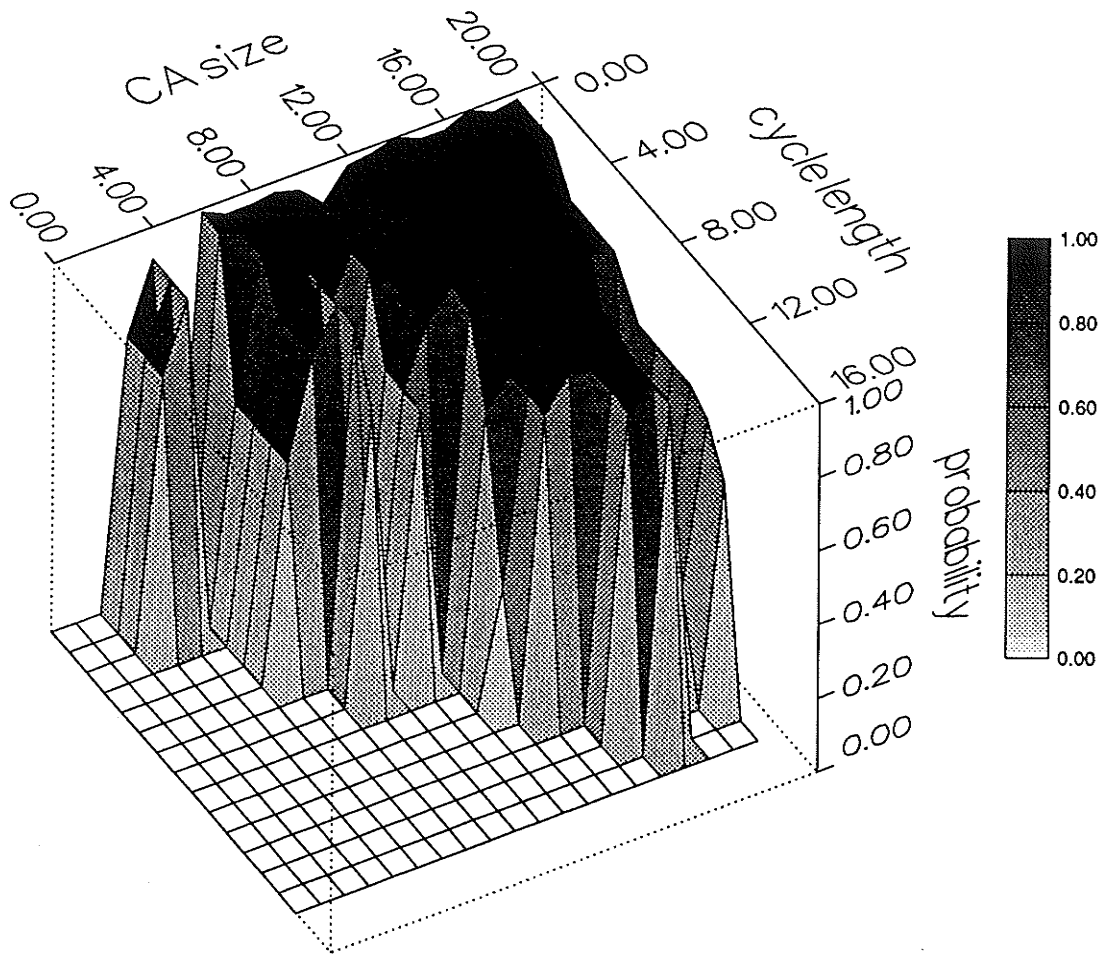


Figure 2.28 : *Cycle length versus probability of entering a cycle of that length for CA rule 30 with cyclic boundary conditions. See appendix B for complete tables.*

boundary conditions the maximum cycle length does not increase nearly so rapidly with cellular automata size and we see that the maximum possible path length becomes nearly constant for large CA rule 30. Thus, the paths for null boundary conditions are much shorter than those provided by cyclic boundary conditions. Therefore, cyclic layouts for CA rule 30 based PRNGs should be used if the length of the sequence is a concern. Another point to note is that there is only one large uncorrelated sequence for each register size. If different uncorrelated sequences are required, for example in multiple Monte Carlo simulations, then the same precautions as when using other algorithmic PRNGs to avoid sequence cross-correlation should be observed (i.e. starting at sufficiently well spaced locations on the cycle to avoid sequence overlap).

As we have already seen, as the length of a rule 30 cellular autmaton increases the maximum possible length of the pseudorandom sequence also increases, but this

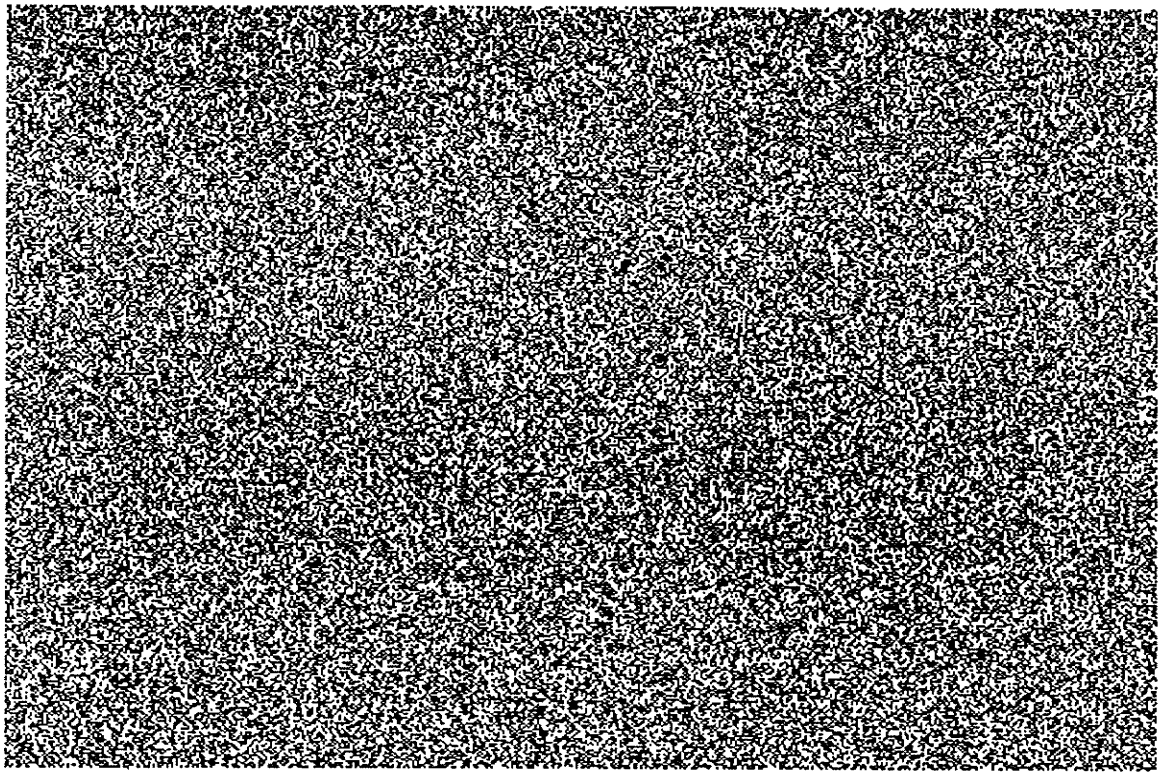


Figure 2.29 : *The raster scan output of a single site in a 32 site rule 45 cellular automaton.*

growth is not monotonic and the seed, or initial state, used in a particular rule 30 cellular automaton will affect the length of the sequence produced. This is not a major problem since a table of *good* seed values, such as those in Table 2.7, can be produced and used in a similar manner to other tables of feedback taps for maximal length LFSR sequences. It is wise to use computer simulation to check the cycle length of the sequence produced, if a different seed value or length than those in Table 2.7 are to be used.

Using Table 2.8 we see that the state transitions for CA rule 30 with cyclic boundary conditions are increasingly dominated by one cycle which is usually much longer than the others. Therefore, an arbitrary starting state has an increasing probability of being in the maximum length cycle or on a path leading to it. From Table 2.8 we also note the varying size and number of the cycles as n increases. More complete results are given in Appendix B. A plot of cycle length versus probability of an arbitrary starting state being in a cycle of that length can be made using the data of Appendix B and is shown in Fig. 2.28. The various curves show the results for differing register lengths. Therefore, if we wish to use an arbitrary starting state and require a probability of 0.95 that a non repeating sequence of length ≥ 1000 will be produced then we need a rule 30 cellular automaton of size ≥ 15. Table 2.8 and Fig. 2.28 only consider CA rule 30 for sizes 4 through 20. Results for larger rule 30 cellular automata are easily derived

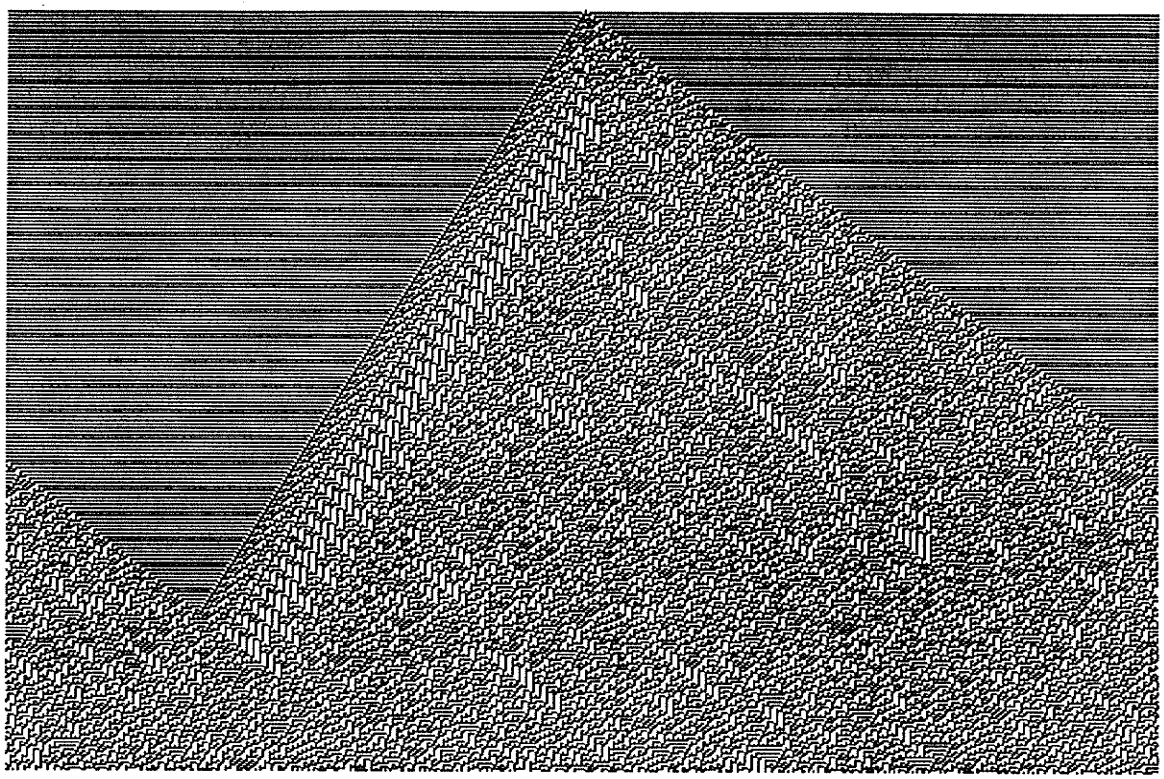


Figure 2.30 : *420 time steps in the state - time diagram of a 500 site rule 45 cellular automaton. Cyclic boundary conditions; initialised with a single nonzero site.*

using a computer program but the run time of these programs becomes quite large for lengths > 20.

2.4.3. CA rule 45

The other autoplectic cellular automaton of interest is rule 45 which is simply

$$a_i(t+1) = a_{i-1}(t) \oplus (a_i(t) \cup \overline{a_{i+1}(t)}) \; . \tag{2.33}$$

The area used by a 30 bit rule 45 cellular automaton is $1.3 \times 10^6 \; \mu m^2$ compared to $1.1 \times 10^6 \; \mu m^2$ for rule 30 and $0.46 \times 10^6 \mu m^2$ for the LFSR. The increased area over CA rule 30 comes from the additional inverter required at each site. CA rule 45 uses about 2.8 times the area of the LFSR but retains the global wiring advantages of CA rule 30 (i.e. nearest neighbour wiring).

As with CA rule 30, it has been investigated extensively and exhibits autoplectic properties in the bit sequence occurring at a single site, $a_i(t)$ [Wolfram1986a]. This can be seen in the raster scan output of a single site, as shown in Fig. 2.29. As well, Figs. 2.30 and 2.31, which show 420 time steps in the state - time diagram of a 500 site rule 45 cellular automaton, further confirm the autoplectic and homoplectic nature of this rule. The single site initialisation of Fig. 2.30 shows that the evolution of CA rule

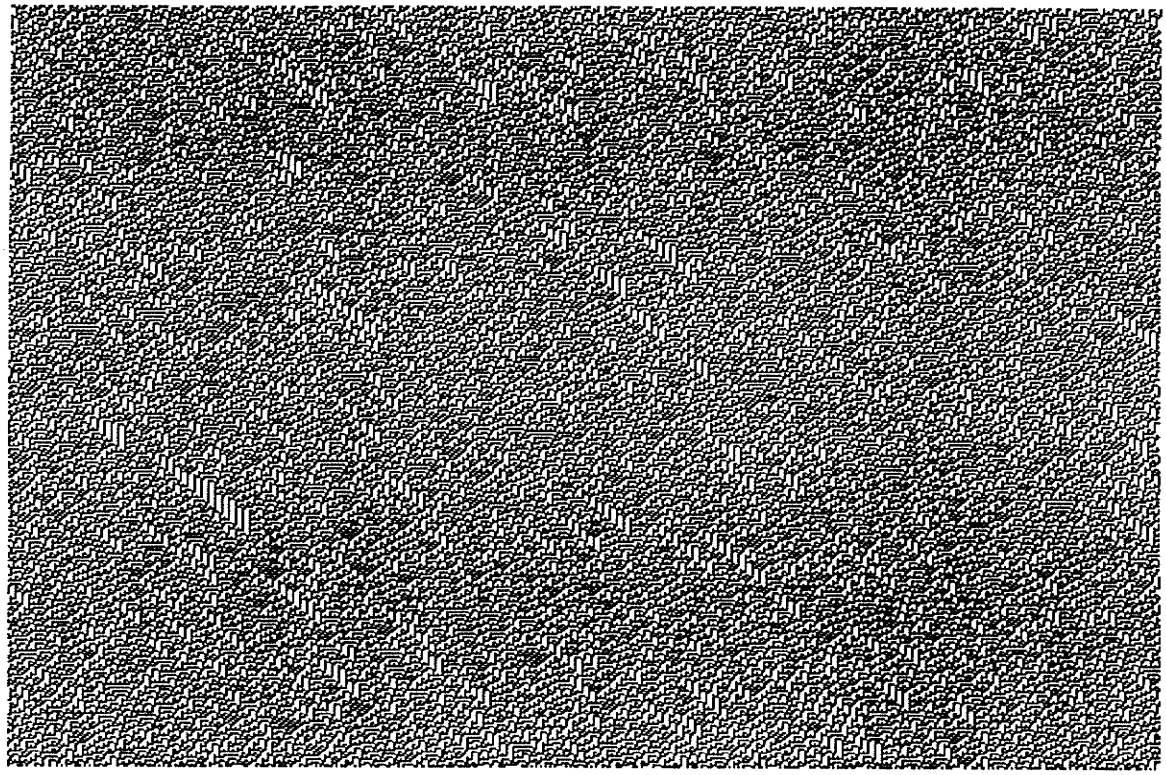


Figure 2.31 : *420 time steps in the state - time diagram of a 500 site rule 45 cellular automaton. Cyclic boundary conditions; random initial state.*

45 is different from CA rule 30 in that sites which have not been reached by the evolution of states alternate between zero and one (the horizontal striped pattern). In addition, there are no triangular shapes but rather, what could be termed threads running throughout the diagram. Perhaps even more importantly, CA rule 45 is very asymmetric with the evolution to the left of Fig. 2.30 being much slower than to the right. Due to these differences we would expect the randomness properties of CA rule 45 to be different from those of CA rule 30.

As with CA rule 30, adjacent sites are somewhat correlated in time. This can be seen in the auto and cross-correlation data of Fig. 2.32. We see that adjacent sites have a cross-correlation of about 0.52 which will cause problems if these two bits are used to generate the same random words. As with CA rule 30, the correlation dies out over a period of time but it takes a larger number of sites. In fact, it takes 13 sites before the correlation falls below 10% and the cross-correlation ridge is visible across the entire word size of Fig. 2.32. As well, the correlation dies out in a very irregular fashion with the cross-correlation ridge rising and falling sharply. Therefore, while CA rule 45 definitely provides advantages over the parallel LFSR, CA rule 30 displays much better cross-correlation properties.

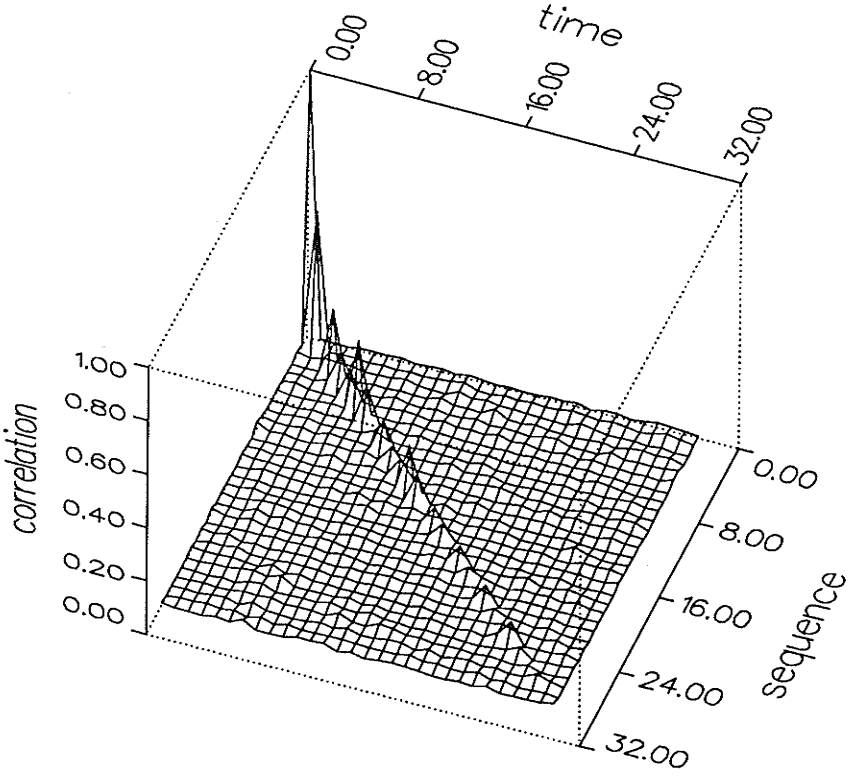


Figure 2.32 : *Auto and cross-correlation of site values in a 30 site rule 45 cellular automaton.*

We can use site and time spacing as with CA rule 30 to try and form a better pseudorandom sequence but we would expect that similar performance to CA rule 30 will require a larger site and time spacing. The results of site spacing for CA rule 45, given in Table 2.9, confirm this suspicion. The average failure metric decreases but not monotonically and for the largest site spacing given in Table 2.9 we see that it has not reached the level of CA rule 30 with $\gamma = 3$. This would indicate that larger site spacing is required to completely remove the bit sequence correlation and reduce the average failure metric to that of CA rule 30 with $\gamma = 4$. In Table 2.10 the test results for CA rule 45 with time spacing are given. As with the site spacing case, a larger time spacing is required with CA rule 45 to obtain similar average failure metrics to those of CA rule 30 with $\beta = 4$. However, from both Tables 2.9 and 2.10 we see that the difference between CA rule 30 and CA rule 45 is quite small, usually less than 2.0, so that the randomness of both CA rules is actually quite close for both site and time spacing.

In deciding which CA rule (i.e. rule 30 or 45) is more suitable for use as a PRNG one must consider several factors including area, randomness, and cycle length. We know that CA rule 45 uses one more inverter than CA rule 30. However, it is possible to avoid this additional inverter by using the q′ output from the storage d flipflop at each site in the cellular automaton, so we will consider the two CA rules to have

Sequence length = 1,000

Test	γ = 0		γ = 1		γ = 2		γ = 3		γ = 4		γ = 5		γ = 6	
mod	10	100	10	100	10	100	10	100	10	100	10	100	10	100
1	84	100	100	100	82	67	50	100	74	76	71	100	100	100
2	21	0	100	100	100	100	74	100	100	78	100	100	100	100
3	84	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100	skip
4	100	skip	79	skip	100	skip	100	skip	100	skip	100	skip	100	skip
5	100	100	100	100	100	81	100	100	100	78	100	100	100	100
6	84	100	100	100	100	100	100	72	100	76	100	100	79	100
7	100	100	100	79	100	100	100	100	100	100	100	100	60	100
8	100	100	77	100	100	100	100	100	100	100	100	100	100	100
9	100	100	100	100	100	100	100	100	100	100	71	100	100	100
10	70	100	100	100	100	100	100	100	100	100	100	100	100	100
11	100	100	72	100	100	100	100	100	100	100	100	100	100	100
12	49	skip	100	skip	82	skip	76	skip	72	skip	71	skip	60	skip
13	100	100	100	100	100	100	100	100	100	100	100	100	100	100
14	100	100	77	100	70	100	100	72	100	78	100	50	100	79
15	skip	100	skip	100	skip	100	skip	78	skip	74	skip	100	skip	100
16	skip	100	skip	100	skip	100	skip	100	skip	78	skip	100	skip	79
17	skip	67	skip	51	skip	100	skip	100	skip	74	skip	100	skip	79
18	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100
19	skip	100	skip	100	skip	81	skip	100	skip	78	skip	100	skip	100
20	67	100	79	56	100	100	100	72	100	100	100	100	39	100
21	63	21	28	100	70	63	78	52	100	78	62	100	85	100
22	0.52		0.03		0.24		0.14		0.12		0.28		0.15	
23	Fail		Fail		Fail		Pass		Pass		Pass		Pass	
24	7	5	5	5	4	5	3	4	2	7	4	2	4	4
25	4.78	4.12	3.88	3.14	2.96	3.08	2.22	2.54	1.54	3.32	2.25	1.50	2.77	1.63

Sequence length = 10,000

Test	γ = 0		γ = 1		γ = 2		γ = 3		γ = 4		γ = 5		γ = 6	
mod	10	100	10	100	10	100	10	100	10	100	10	100	10	100
1	72	100	70	100	100	49	100	100	100	77	74	83	100	100
2	0	0	70	100	80	100	100	100	100	100	75	100	100	100
3	0	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100	skip
4	56	skip	71	skip	100	skip	100	skip	100	skip	68	skip	100	skip
5	56	77	100	100	100	80	79	100	100	100	100	100	100	79
6	100	77	100	100	100	45	100	100	100	100	100	100	100	100
7	100	100	71	100	100	100	100	100	100	100	75	100	100	100
8	72	100	100	100	100	100	73	100	78	100	100	100	74	71
9	100	79	49	70	100	100	100	100	77	100	100	100	100	100
10	100	100	100	100	100	100	48	100	100	100	83	100	100	100
11	100	100	78	100	100	100	100	100	100	100	100	100	100	100
12	100	skip	100	skip	100	skip	73	skip	100	skip	100	skip	100	skip
13	100	100	100	100	100	100	100	100	100	100	100	100	100	76
14	100	100	100	100	80	100	100	81	77	100	100	100	100	71
15	skip	72	skip	78	skip	69	skip	73	skip	69	skip	100	skip	79
16	skip	100	skip	100	skip	100	skip	100	skip	76	skip	100	skip	100
17	skip	77	skip	100	skip	75	skip	100	skip	100	skip	100	skip	100
18	skip	100	skip	100	skip	69	skip	67	skip	77	skip	100	skip	100
19	skip	100	skip	100	skip	80	skip	100	skip	100	skip	100	skip	100
20	72	72	100	71	100	100	79	100	69	100	100	100	74	74
21	0	21	81	30	100	100	40	81	100	69	100	43	50	79
22	0.52		0.03		0.24		0.14		0.12		0.28		0.15	
23	Fail		Fail		Fail		Pass		Pass		Pass		Pass	
24	8	7	5	4	4	5	4	3	3	3	3	2	3	4
25	6.72	5.25	4.10	3.51	2.40	4.33	3.08	1.98	1.99	2.32	2.25	1.74	2.02	2.71

Table 2.9: *Random number test results for CA rule 45 with various site spacing values, γ.*

Sequence length = 1,000

Test	β = 0		β = 1		β = 2		β = 3		β = 4		β = 5		β = 6	
mod	10	100	10	100	10	100	10	100	10	100	10	100	10	100
1	81	100	72	100	100	100	100	100	100	100	100	100	100	100
2	22	0	100	100	100	100	46	100	100	100	100	100	100	75
3	81	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100	skip
4	100	skip	100	skip	100	skip	74	skip	76	skip	100	skip	65	skip
5	100	100	100	76	81	100	100	100	100	100	100	100	75	100
6	81	100	100	100	100	72	79	100	100	100	100	100	100	100
7	100	100	72	100	72	100	100	100	100	100	77	100	100	100
8	100	100	100	72	100	72	100	100	100	100	100	100	100	100
9	100	100	100	100	81	100	100	100	76	100	100	100	100	65
10	70	100	72	100	76	100	100	74	100	100	100	77	100	40
11	100	100	100	100	100	100	100	79	100	100	100	100	100	100
12	48	skip	76	skip	100	skip	72	skip	100	skip	100	skip	100	skip
13	100	100	100	100	100	100	100	74	100	100	100	100	100	100
14	100	100	100	100	71	100	100	100	100	73	71	77	100	100
15	skip	100	skip	72	skip	100	skip	100	skip	73	skip	100	skip	100
16	skip	100	skip	100	skip	100	skip	100	skip	100	skip	71	skip	83
17	skip	71	skip	100	skip	100	skip	100	skip	73	skip	100	skip	100
18	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100
19	skip	100	skip	100	skip	100	skip	100	skip	100	skip	77	skip	100
20	71	100	100	76	100	100	79	100	100	78	100	100	100	100
21	59	22	78	76	57	53	74	53	76	49	29	77	100	83
22	0.52		0.03		0.24		0.14		0.12		0.28		0.15	
23	Fail		Fail		Fail		Pass		Pass		Pass		Pass	
24	7	5	5	5	5	5	4	4	4	3	3	5	2	3
25	4.87	4.07	3.30	3.28	3.62	3.03	2.76	2.20	1.72	2.54	2.23	2.21	1.60	2.54

Sequence length = 10,000

Test	β = 0		β = 1		β = 2		β = 3		β = 4		β = 5		β = 6	
mod	10	100	10	100	10	100	10	100	10	100	10	100	10	100
1	70	101	68	74	101	97	74	97	54	79	95	70	79	79
2	78	78	74	95	101	86	16	23	88	81	95	100	85	93
3	55	skip	101	skip	101	skip	23	skip	94	skip	100	skip	100	skip
4	91	skip	95	skip	101	skip	62	skip	81	skip	100	skip	100	skip
5	91	95	101	101	101	84	62	85	100	100	100	100	100	95
6	101	95	101	81	101	94	100	85	100	100	100	78	100	100
7	101	101	101	101	101	101	100	100	100	100	100	100	100	100
8	94	101	101	101	82	101	74	100	100	100	76	100	71	100
9	101	73	86	101	101	101	97	78	100	95	92	100	79	100
10	101	101	101	101	101	101	100	89	100	100	100	100	100	61
11	79	101	95	101	95	101	100	96	82	100	82	100	100	77
12	79	skip	101	skip	86	skip	100	skip	94	skip	100	skip	93	skip
13	101	101	81	101	63	101	100	100	100	100	96	100	92	100
14	101	101	95	101	81	84	100	100	81	100	100	100	97	95
15	skip	95	skip	101	skip	101	skip	84	skip	100	skip	100	skip	100
16	skip	101	skip	81	skip	77	skip	100	skip	100	skip	70	skip	97
17	skip	95	skip	80	skip	95	skip	85	skip	88	skip	73	skip	100
18	skip	101	skip	95	skip	101	skip	100	skip	88	skip	100	skip	100
19	skip	101	skip	101	skip	101	skip	97	skip	100	skip	100	skip	79
20	70	94	81	101	94	84	73	68	95	100	77	100	77	78
21	61	59	38	101	88	70	13	36	87	94	45	65	63	58
22	0.52		0.03		0.24		0.14		0.12		0.28		0.15	
23	Fail		Fail		Fail		Pass		Pass		Pass		Pass	
24	8	7	5	3	4	6	7	6	4	3	5	3	5	5
25	4.43	3.24	2.97	2.01	3.19	3.40	5.03	3.79	2.46	1.77	2.44	2.43	2.65	2.90

Table 2.10: *Random number test results for CA rule 45 with various time spacing values, β.*

N	Cycles	Frac. longest.
4	1x2	1.00
5	1x30, 1x2	0.94
6	1x18, 1x3, 1x2, 3x1	0.84
7	1x126, 1x2	0.98
8	1x32, 2x24, 1x16, 2x4, 1x2	0.13
9	1x504, 1x3, 1x2, 3x1	0.98
10	1x403, 4x60, 2x15, 1x2	0.70
11	1x979, 1x935, 1x66, 1x11, 11x5, 1x2	0.48
12	1x240, 1x156, 1x84, 12x24, 1x18, 12x12, 1x3, 1x2, 3x1	0.06
13	1x1105, 1x676, 13x443, 1x156, 1x130, 4x78, 1x39, 1x13, 1x2	0.13
14	1x2198, 7x534, 3x392, 2x168, 1x126, 2x42, 1x2	0.52
15	1x6820, 1x4920, 1x2820, 1x2340, 3x120, 21x60, 15x32, 4x30, 1x3, 2x2,3x1	0.21
16	1x2816, 1x976, 1x848, 4x700, 4x556, 4x296, 1x208, 2x144, 17x48, 1x32, 2x24, 1x16, 4x2, 2x1	0.06
17	1x78812, 1x32912, 1x6052,1x4845, 1x867,1x816, 7x408, 4x204, 1x102, 1x2	0.60
18	1x8787, 1x8168, 2x7812, 3x3756, 1x504, 12x72, 90x36, 6x21, 81x18, 1x3, 2x2, 3x1	0.18
19	1x183920, 1x158080, 1x149425, 1x15371, 1x3458, 1x1653, 1x1425, 5x912, 10x456, 1x361, 1x228, 10x114, 1x95, 1x2	0.35
20	1x142580, 4x14265, 5x9112, 1x4260, 1x110, 1x480, 1x430, 4x280, 5x252, 9x240, 5x236, 72x120, 166x60, 5x30, 2x15, 1x2	0.48

Table 2.11: *Cycles lengths for various size rule 45 cellular automata and the fraction of all states leading to the longest cycle.*

equivalent area. As discussed above, CA rules 30 and 45 have nearly equivalent randomness in terms of the word wide sequence that is generated. Another metric which can be used to measure the randomness of a bit sequence (i.e. such as that occurring from each site in the cellular automaton) is the entropy of the sequence. The entropy provides a characterisation as to the number of possible sequences that may occur. Here we define two entropy measures: the topological entropy[Wolfram1984c]

$$s = \lim_{n\to\infty} \frac{1}{n} \log_2 N(n) \quad ; \tag{2.34}$$

and the measure entropy

$$s_\mu = \lim_{n\to\infty} \frac{-1}{n} \sum_{i=1}^{2^n} p_i \log_2 p_i \quad ; \tag{2.35}$$

where

$N(n)$ = the number of distinct length n blocks in these sequences.

p_i = the probability of sequence i appearing.

If we consider the sequences to be messages on a communication channel then the entropies correspond to the channel capacity and Shannon information content,

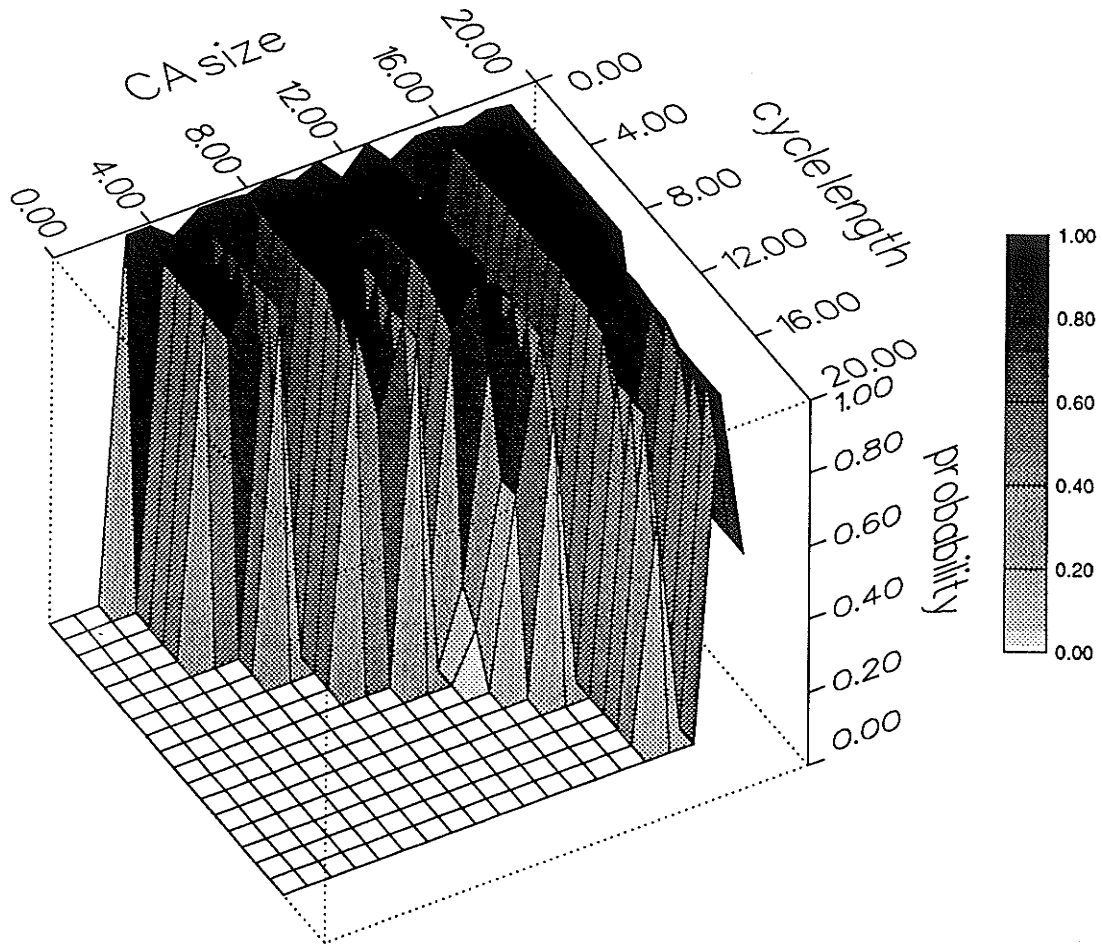


Figure 2.33 : *Cycle length versus probability of entering a cycle of that length for CA rule 45 with cyclic boundary conditions. See appendix B for complete tables.*

respectively [Wolfram1986a]. Therefore, for a random sequence generator we expect all sequences to be equally likely and so, both entropies should be maximal, i.e.

$$s = s_{\mu} = 1 \quad . \tag{2.36}$$

This measure of randomness is very similar to the equidistribution and serial tests described previously. Wolfram [Wolfram1986a] has found that the entropy of the bit sequence from each site in a rule 45 cellular automaton is slightly smaller than that of CA rule 30 so CA rule 45 must have some repeating blocks in the bit sequence from each site as compared to CA rule 30. Incidentally, Wolfram has also found that the entropy from a single site in a rule 30 cellular automaton is maximal (i.e. equals one). Therefore, as with the site and time spacing measures, CA rule 30 is slightly more random than CA rule 45.

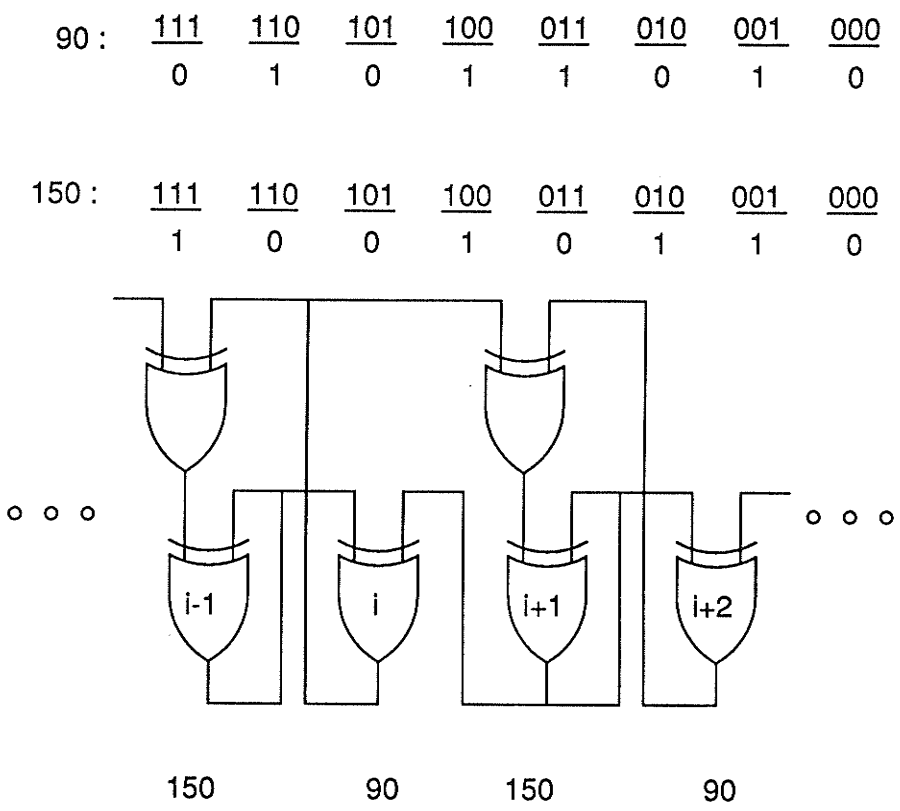


Figure 2.34 : *A hybrid cellular automaton using CA rules 90 and 150 at alternating sites.*

Another aspect which can be considered is the propagation speed of the *randomness* to the left and right in both CA rules 30 and 45. This can be found by considering the *difference pattern* produced by subtracting two randomly initialised cellular automata where the seeds differ in only one bit. The difference pattern looks similar to the single initialised site state - time diagrams of Figs. 2.17 and 2.30. The slope of the difference pattern to the left and right shows the information transmission of the differing bit over the entire cellular automaton. This can also be thought of as rate of spread of *randomness* over the cellular automaton. The left and right slopes yield the left and right Lyapunov exponents, λ_L and λ_R, respectively for cellular automaton's evolution [Wolfram1984c], [Packard1985a]. Both CA rule 30 and 45 have $\lambda_R = 1.0$ but for CA rule 30 it can be shown that $\lambda_L = 0.2428 \pm 0.0003$ while for CA rule 45 $\lambda_L = 0.1724 \pm 0.0004$ [Wolfram1986a]. Therefore, *randomness* in CA rule 30 spreads to the left at a slope of about 28% more than in CA rule 45.

Finally, we must consider the cycle lengths of CA rules 30 and 45. Table 2.11 is a list of all cycles for CA rule 45 as well as the percentage of all states which are members of, or are on paths leading to, the longest cycle. Figure 2.33 uses the data of Appendix B for CA rule 45 to produce a plot analogous to that of Fig. 2.28 for CA rule 30. When these two plots are compared we see that as the length of CA rule 45 is increased it is not dominated by only one large cycle but may have several large

Length	Construction	Cycle length
4	0101	15
5	11001	31
6	010101	63
7	1101010	127
8	11010101	255
9	110010101	511
10	0101010101	1,023
11	11010101010	2,047
12	010101010101	4,095
13	1100101010100	8,191
14	01111101111110	16,383
15	100100010100001	32,767
16	1101010101010101	65,535
17	01111101111110011	131,071
18	010101010101010101	262,143
19	0110100110110001001	524,867
20	11110011101101111111	1,048,575
21	011110011000001111011	2,097,151
22	0101010101010101010101	4,194,303
23	11010111001110100011010	8,388,607
24	111111010010110101010110	16,777,213
25	101111010101010011110010 0	33,554,431
26	010110101101000101110110 00	67,108,863
27	000011111000001100100001101	134,217,727
28	0101010101010101010101010101	268,435,455

Table 2.12: *Hybrid constructions necessary to achieve a cellular automaton with maximal cycle length. Here* 1 *refers to CA rule 150. Hence, a length* 5 *maximal length hybrid would be constructed by having CA rules 90 and 150 in the following order,* 150, 150, 90, 90, 150. *It should be noted that for many lengths there are several CA rule 90 and 150 hybrid constructions which will yield maximal length cycles. Maximal cycle length hybrid cellular automata exist for lengths larger than 28 but must be found using computer simulation.*

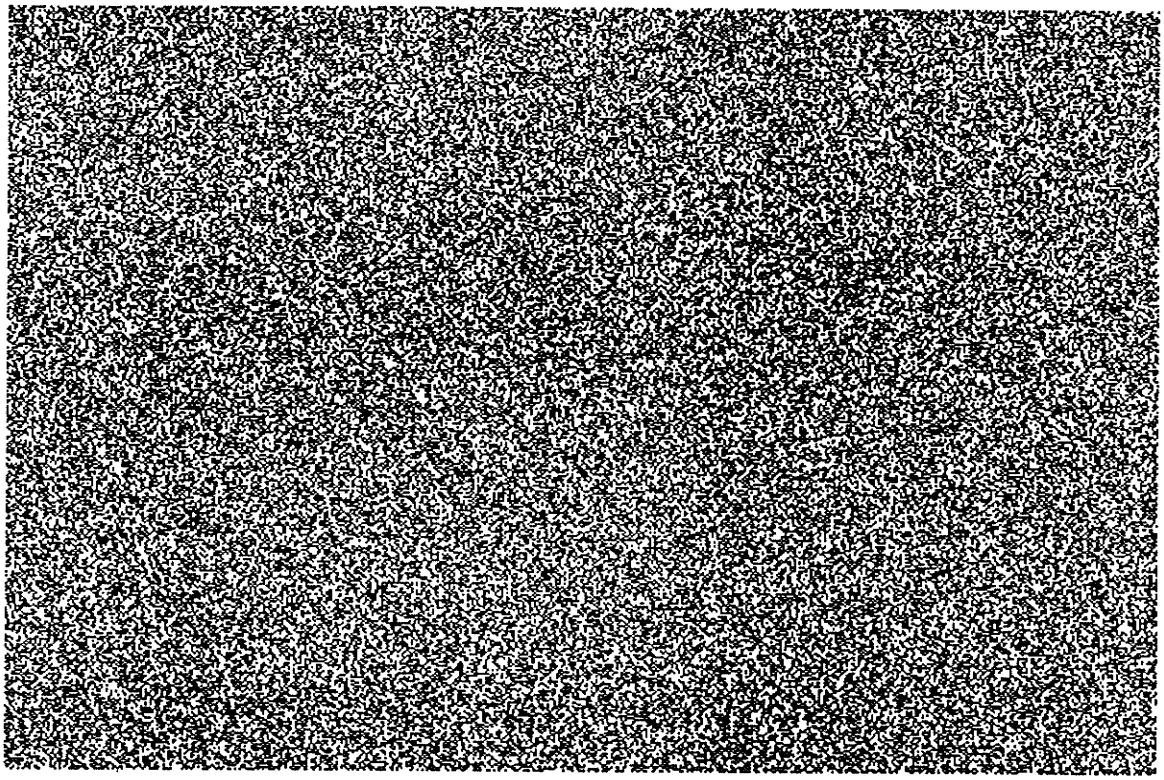


Figure 2.35 : *Output of a single site in a 30 site rule 90 and 150 hybrid cellular automaton.*

cycles. This severely reduces the probability of arbitrarily starting in, or on a path leading to, the largest cycle. However, the probability of starting in a nonrepeating sequence of a length greater than some fixed value is better in CA rule 45 since the probability of being in one of the larger cycles of CA rule 45 is greater than entering the one largest cycle of CA rule 30. Notice that the longest cycle in CA rule 45 is usually several times that of CA rule 30. Therefore, while CA rule 30 has better randomness properties than CA rule 45, CA rule 45 provides much larger cycle lengths. In this work we are mainly concerned with generating good pseudorandom sequences and so, we will consider the CA rule 30 based PRNG to be better than the CA rule 45 based PRNG. However, if for some application cycle length becomes an important consideration then the CA rule 45 based PRNG should be seriously considered. Another point to be noted is that for odd cycle lengths no state has more than one predecessor since the state transition diagram for odd length CA rule 45 contains only cycles and no paths leading to cycles. Finally, it can be shown that similar behaviour to CA rule 30 with null boundary conditions is exhibited by CA rule 45 with null boundary conditions (i.e. very short cycle lengths).

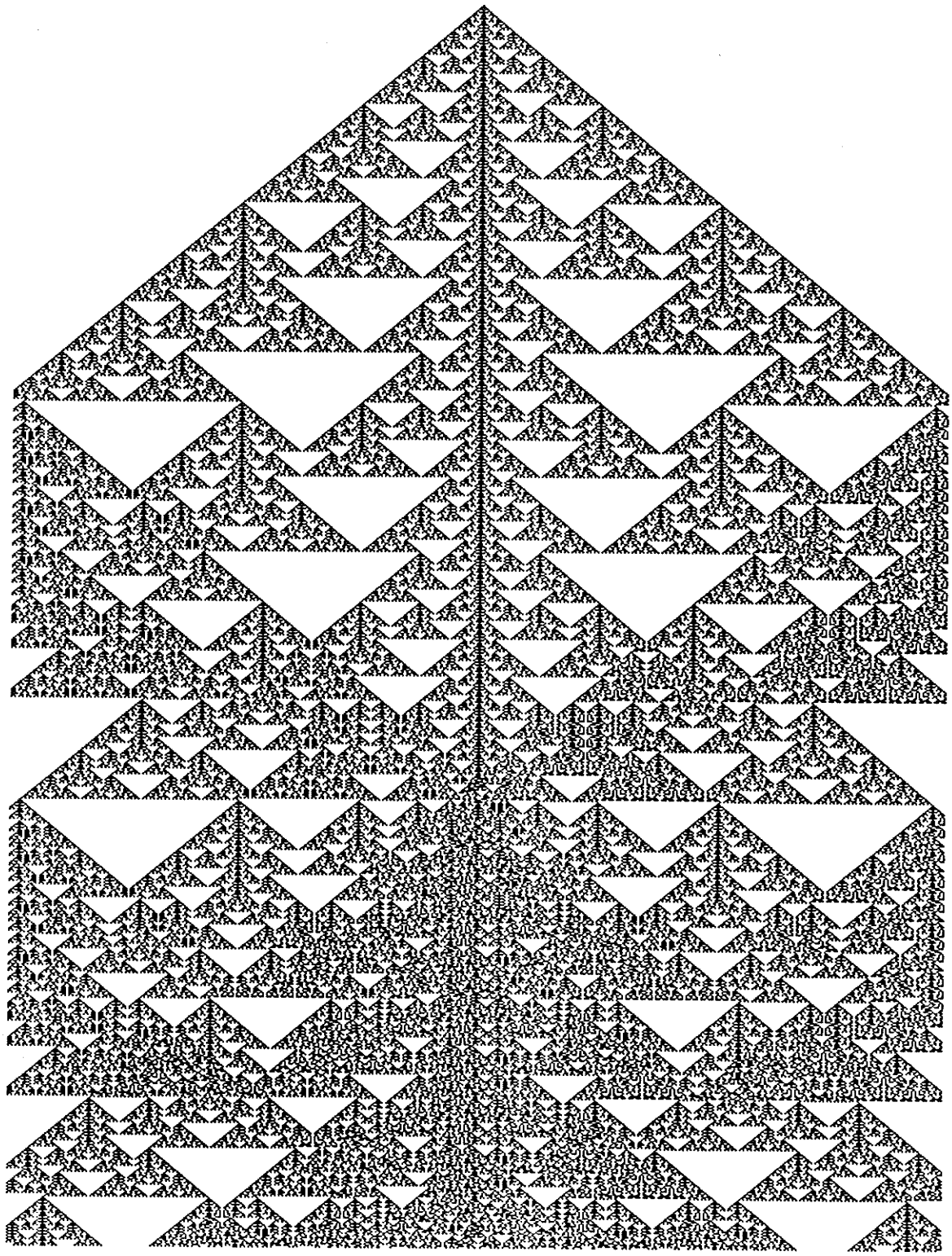


Figure 2.36 : *800 time steps in the state - time diagram of a 498 site rule 90 and 150 hybrid cellular automaton. Null boundary conditions; initialised with a single nonzero site.*

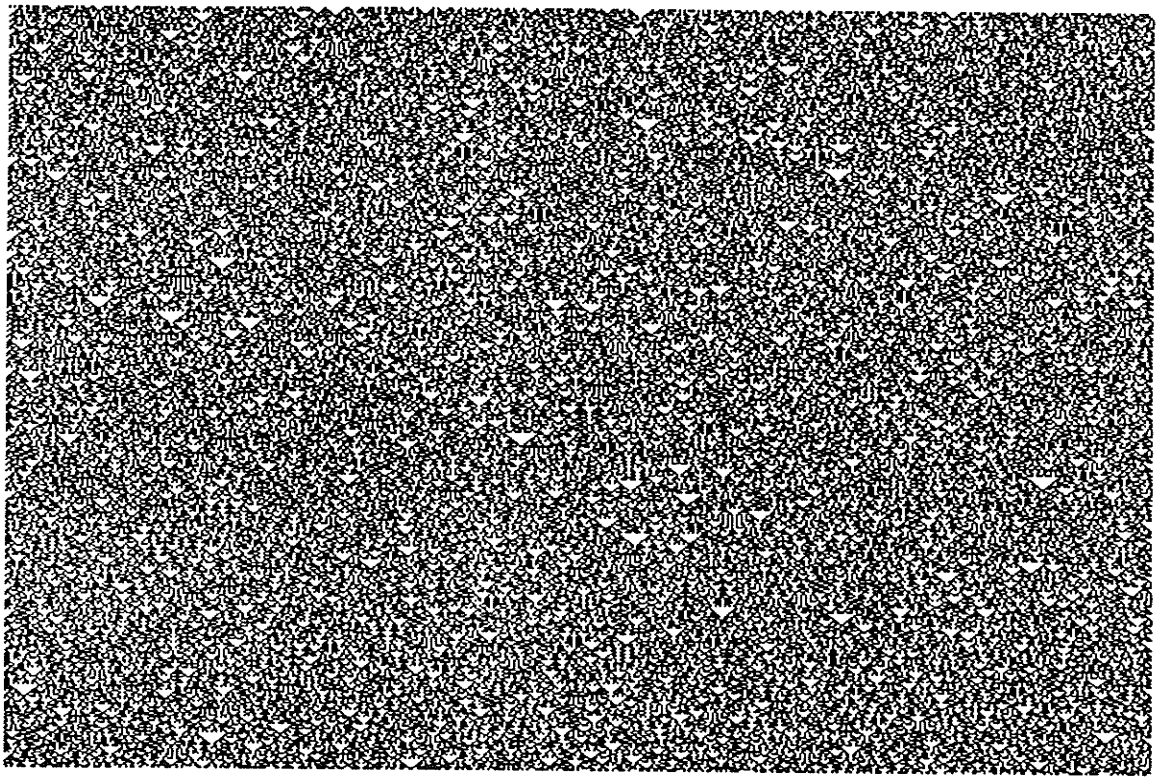


Figure 2.37 : *420 time steps in the state - time diagram of a 498 site rule 90 and 150 hybrid cellular automaton. Null boundary conditions; random initial state.*

2.4.4. Homoplectic CA-Based PRNGs

To generate pseudorandom numbers in parallel using cellular automata it would appear that using cellular automata which exhibit either homoplectic or autoplectic behaviour would suffice. However, homoplectic cellular automata only output pseudorandom patterns for certain input states. For example, consider the first $N/2$ sites having a zero value and the next $N/2$ sites having a one value. This seed would obey all the requirements of a random initial state since $p(0) = p(1) = 0.5$ but would most certainly yield an output which would be decidedly nonrandom. In fact, the author has found it difficult to find starting states for some homoplectic class 3 CA rules which will yield random sequences. Thus, the homoplectic nature of some class 3 CA rules is only over an ensemble of different starting or seed values. On the other hand, CA rules exhibiting autoplectic behaviour will produce a pseudorandom output independent of the seed value. Therefore, autoplectic behaviour is more desirable in a PRNG since we need not be concerned about whether or not the seed value will produce pseudorandom behaviour. Thus, for the purposes of this work, homoplectic CA rules will not be considered appropriate and will no longer be considered.

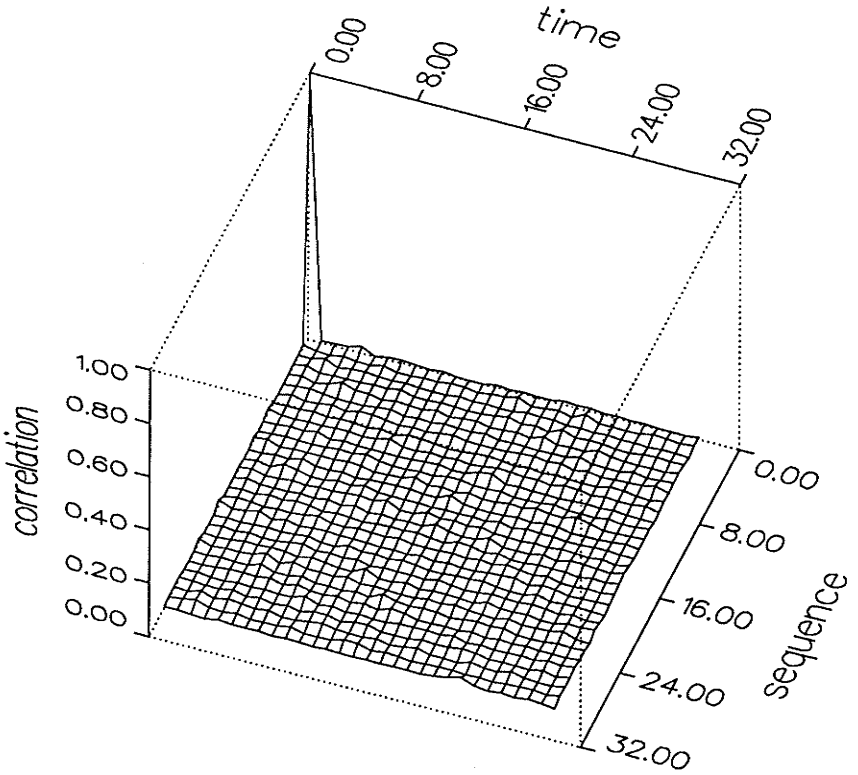


Figure 2.38 : *Auto and cross-correlation of site values in a 30 site rule 90 and 150 hybrid cellular automaton.*

2.4.5. Hybrid Cellular Automata

A cellular automaton which yields a maximal length binary sequence from each site, like the LFSR, is the rule 90 and 150 *hybrid* cellular automaton of Fig. 2.34 which was initially proposed by Pries [Pries1986]. Here the surprising combination of CA rule 90

$$a_i(t+1) = a_{i-1}(t) \oplus a_{i+1}(t) \tag{2.37}$$

and CA rule 150

$$a_i(t+1) = a_{i-1}(t) \oplus a_i(t) \oplus a_{i+1}(t) \tag{2.38}$$

both of which are simple linear rules, yields maximum length cycles (i.e. $2^n - 1$, n = length of the cellular automaton) . It has been conjectured, based upon computer simulation, that to achieve maximal length cycles the length of the cellular automaton is subject to the constraints that

$$n \bmod 2 = 0 \ , \ n \bmod 3 \neq 2 \tag{2.39}$$

and it must have null boundary conditions. However, these computer simulations where only carried out up to length 12 [Pries1987]. Computer simulations performed for this work show that a further restriction of $n \bmod 8 \neq 0$ must be added for

Sequence length = 1,000

Test	γ = 0		γ = 1		γ = 2		γ = 3		γ = 4		γ = 5		γ = 6	
mod	10	100	10	100	10	100	10	100	10	100	10	100	10	100
1	100	75	49	75	100	71	100	100	100	75	70	100	100	100
2	100	100	100	100	100	100	75	100	81	100	100	75	100	100
3	100	skip	100	skip	81	skip	74	skip	69	skip	100	skip	100	skip
4	100	skip	74	skip	81	skip	100	skip	44	skip	100	skip	100	skip
5	100	100	49	100	100	100	100	100	100	69	100	100	100	100
6	100	100	100	100	100	100	74	100	100	100	100	100	100	100
7	100	100	100	100	100	100	100	73	100	100	100	100	100	100
8	100	100	100	100	100	100	100	100	100	100	100	100	100	100
9	79	100	100	100	100	100	75	100	100	100	100	100	100	72
10	100	100	100	81	76	100	75	75	75	100	70	100	100	100
11	100	100	100	100	76	100	75	100	100	100	70	100	100	100
12	79	skip	74	skip	100	skip	78	skip	100	skip	100	skip	72	skip
13	100	75	100	100	100	100	100	100	100	100	100	100	100	100
14	100	71	100	100	100	100	100	100	56	100	73	100	100	100
15	skip	100	skip	100	skip	100	skip	75	skip	100	skip	100	skip	100
16	skip	100	skip	100	skip	100	skip	100	skip	75	skip	100	skip	80
17	skip	100	skip	100	skip	100	skip	74	skip	100	skip	100	skip	100
18	skip	100	skip	100	skip	100	skip	49	skip	100	skip	100	skip	100
19	skip	100	skip	100	skip	100	skip	74	skip	75	skip	70	skip	72
20	100	100	19	100	71	100	74	75	100	100	100	100	100	100
21	100	54	56	100	71	57	78	0	44	81	75	100	80	27
22	0.12		0.05		0.05		0.12		0.12		0.05		0.05	
23	Fail		Fail		Fail		Pass		Pass		Pass		Pass	
24	4	5	6	2	3	2	5	6	3	2	3	1	1	3
25	2.42	3.25	3.79	1.44	2.44	1.72	3.22	4.05	2.31	1.25	1.42	0.55	0.48	1.49

Sequence length = 10,000

Test	γ = 0		γ = 1		γ = 2		γ = 3		γ = 4		γ = 5		γ = 6	
mod	10	100	10	100	10	100	10	100	10	100	10	100	10	100
1	0	0	100	100	100	100	100	100	100	100	72	100	70	100
2	0	0	100	100	100	100	100	62	100	100	100	100	67	100
3	0	skip	100	skip	100	skip	100	skip	70	skip	100	skip	80	skip
4	0	skip	100	skip	100	skip	100	skip	70	skip	100	skip	100	skip
5	0	0	100	100	100	100	100	100	70	100	100	100	50	100
6	0	28	100	100	100	100	41	100	100	100	100	80	100	100
7	20	83	100	100	100	100	100	100	77	100	100	100	100	100
8	48	48	100	79	100	100	100	100	77	100	100	100	100	100
9	20	48	100	100	100	100	79	100	100	100	100	63	67	100
10	65	48	100	100	85	100	100	100	100	70	100	100	67	100
11	37	65	100	100	58	100	100	100	100	77	100	100	100	100
12	0	skip	76	skip	100	skip	100	skip	100	skip	100	skip	100	skip
13	100	100	100	100	100	76	100	79	100	100	100	100	100	100
14	17	0	100	100	100	85	100	100	100	77	100	100	70	100
15	skip	20	skip	100	skip	100	skip	41	skip	77	skip	80	skip	83
16	skip	0	skip	100	skip	100	skip	100	skip	100	skip	100	skip	83
17	skip	0	skip	100	skip	100	skip	79	skip	72	skip	100	skip	100
18	skip	0	skip	100	skip	73	skip	41	skip	77	skip	100	skip	100
19	skip	0	skip	100	skip	76	skip	62	skip	77	skip	100	skip	83
20	0	0	100	76	100	100	100	100	77	100	100	100	70	100
21	17	28	79	67	100	58	62	80	42	81	65	52	100	47
22	0.05		0.05		0.05		0.05		0.12		0.05		0.05	
23	Fail		Fail		Fail		Pass		Pass		Pass		Pass	
24	16	17	2	2	3	3	2	4	4	5	1	3	4	3
25	13.8	14.3	1.45	1.78	1.57	2.32	1.18	2.56	2.17	1.92	0.63	1.25	2.59	1.04

Table 2.13: *Random number test results for the rule 90 and 150 hybrid with various site spacing values, γ.*

Sequence length = 1,000

Test	β = 0		β = 1		β = 2		β = 3		β = 4		β = 5		β = 6	
mod	10	100	10	100	10	100	10	100	10	100	10	100	10	100
1	100	82	100	69	44	0	100	68	100	82	57	0	100	100
2	100	100	75	100	0	0	100	68	100	100	0	0	43	66
3	100	skip	75	skip	100	skip	100	skip	100	skip	75	skip	100	skip
4	100	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100	skip
5	100	100	100	100	100	81	77	100	100	100	100	25	100	100
6	100	100	100	100	100	100	100	100	100	100	100	100	100	100
7	100	100	100	100	67	100	100	100	100	100	100	100	72	100
8	100	100	100	100	100	100	100	100	100	100	100	100	100	100
9	68	100	100	100	100	100	100	100	69	100	100	100	85	100
10	100	100	100	100	100	100	68	100	82	100	75	100	77	100
11	100	100	100	100	100	100	100	100	100	100	100	100	100	100
12	68	skip	81	skip	100	skip	100	skip	82	skip	57	skip	66	skip
13	100	82	100	100	100	100	100	100	100	100	100	100	100	100
14	100	75	100	100	100	100	100	100	100	79	100	100	100	100
15	skip	100	skip	100	skip	100	skip	100	skip	69	skip	100	skip	100
16	skip	100	skip	100	skip	100	skip	100	skip	69	skip	100	skip	100
17	skip	100	skip	75	skip	75	skip	100	skip	100	skip	100	skip	77
18	skip	100	skip	100	skip	48	skip	100	skip	100	skip	46	skip	100
19	skip	100	skip	100	skip	19	skip	72	skip	100	skip	71	skip	100
20	100	100	100	100	25	0	100	72	82	79	100	25	77	51
21	100	50	100	100	19	52	68	100	70	18	57	25	77	62
22	0.05		0.05		0.18		0.11		0.05		0.20		0.11	
23	Fail		Fail		Fail		Pass		Pass		Pass		Pass	
24	3	4	2	2	6	7	2	2	3	3	6	7	4	2
25	1.64	2.11	1.69	1.56	4.45	6.25	0.87	1.20	1.15	2.04	2.79	5.08	2.03	1.44

Sequence length = 10,000

Test	β = 0		β = 1		β = 2		β = 3		β = 4		β = 5		β = 6	
mod	10	100	10	100	10	100	10	100	10	100	10	100	10	100
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	skip	0	skip	0	skip	0	skip	0	skip	0	skip	0	skip
4	0	skip	0	skip	0	skip	0	skip	0	skip	0	skip	0	skip
5	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	0	26	28	22	21	79	20	80	48	26	0	47	26	54
7	26	76	28	78	18	50	69	49	100	48	0	0	46	74
8	52	52	0	78	18	32	80	80	100	48	0	0	46	100
9	26	52	0	50	50	0	31	80	22	48	0	23	26	80
10	76	52	78	78	32	0	60	80	100	48	0	0	20	100
11	50	76	78	78	0	0	80	80	74	22	0	0	74	100
12	0	skip	50	skip	50	skip	31	skip	0	skip	52	skip	28	skip
13	100	100	100	100	50	82	100	100	100	100	77	77	100	100
14	24	0	0	0	29	0	0	0	26	0	25	0	0	0
15	skip	26	skip	22	skip	0	skip	40	skip	48	skip	0	skip	26
16	skip	0	skip	0	skip	0	skip	0	skip	0	skip	0	skip	0
17	skip	0	skip	0	skip	0	skip	0	skip	0	skip	0	skip	0
18	skip	0	skip	0	skip	32	skip	20	skip	48	skip	0	skip	52
19	skip	0	skip	0	skip	0	skip	20	skip	0	skip	0	skip	20
20	0	0	22	0	0	0	20	0	52	0	0	0	0	0
21	24	26	0	28	0	0	20	80	26	0	24	0	26	0
22	0.05		0.05		0.18		0.11		0.05		0.17		0.11	
23	Fail		Fail		Fail		Pass		Pass		Pass		Pass	
24	16	17	16	18	15	18	12	13	10	16	16	17	13	12
25	13.2	14.1	13.2	13.7	14.3	16.3	10.9	10.9	9.5	13.6	14.2	16.5	12.1	10.9

Table 2.14: *Random number test results for the rule 90 and 150 hybrid with various time spacing values, β.*

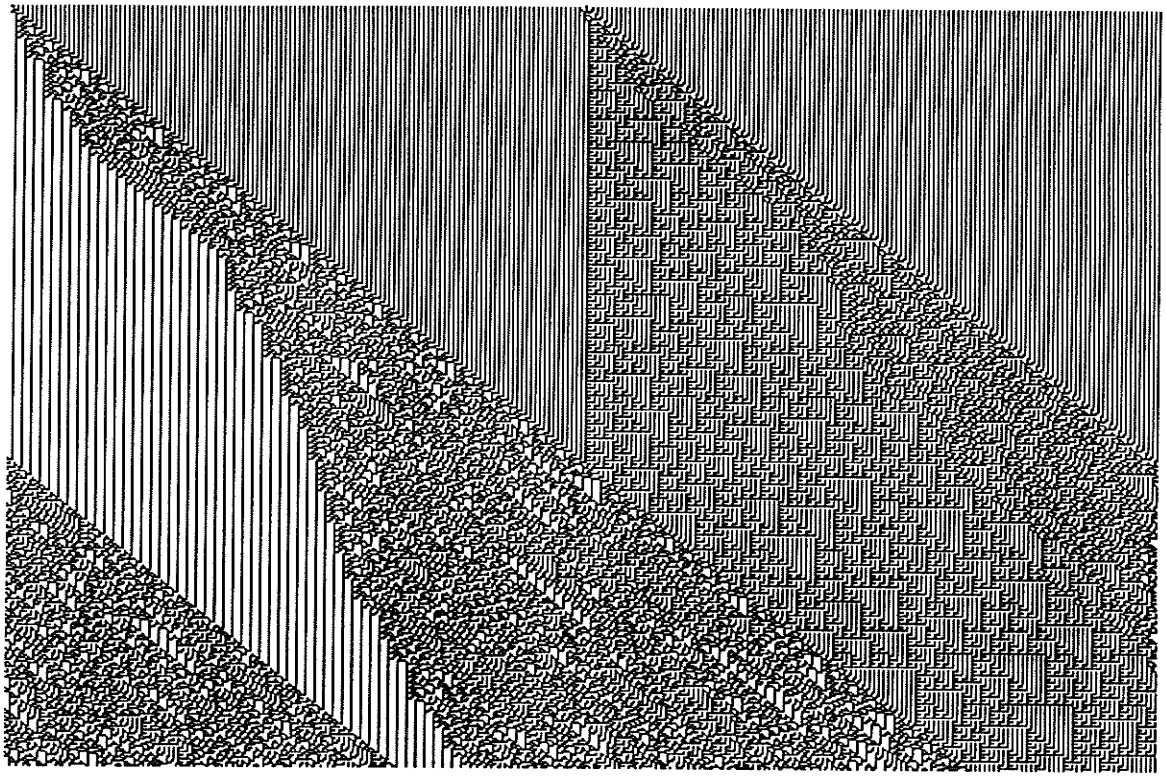


Figure 2.39 : *420 time steps in the state - time diagram of a 500 site rule 30 and 45 hybrid. Cyclic boundary conditions; initialised with a single nonzero site.*

length 16. Furthermore, for longer lengths, ≥ 30, it would appear that even more constraints are required. However, further computer simulation shows that if other hybrid constructions using these two CA rules are utilised then it is possible to form a cellular automaton with maximal cycle length for any desired length. For example, consider a hybrid of length 16. A maximal length cycle can be formed by simply twinning CA rule 150 at one end of the automaton (i.e. rather than having 90, 150, 90, 150, 90, 150, $\cdots$ use 150, 150, 90, 150, 90, 150, $\cdots$). A table indicating the hybrid construction necessary to achieve a cellular automaton with maximal cycle length is given in Table 2.12.

The output of a single site in the rule 90 and 150 hybrid yields a binary sequence as shown in the raster scan of Fig. 2.35. The rule 90 and 150 hybrid method makes effective use of the cellular automaton since all possible outputs are generated. This hybrid is also somewhat autoplectic since a regular starting pattern eventually leads to sequences which closely resemble a pseudorandom sequence. In Fig. 2.36 800 time steps in the state - time diagram of a 498 site rule 90 and 150 hybrid with a simple initial state is shown. Note that the regular pattern dies out as the hybrid evolves in time. However, unlike CA rule 30, the rule 90 and 150 hybrid displays symmetry between the left and right sides of the figure. Figure 2.37 shows the evolution of the rule 90 and

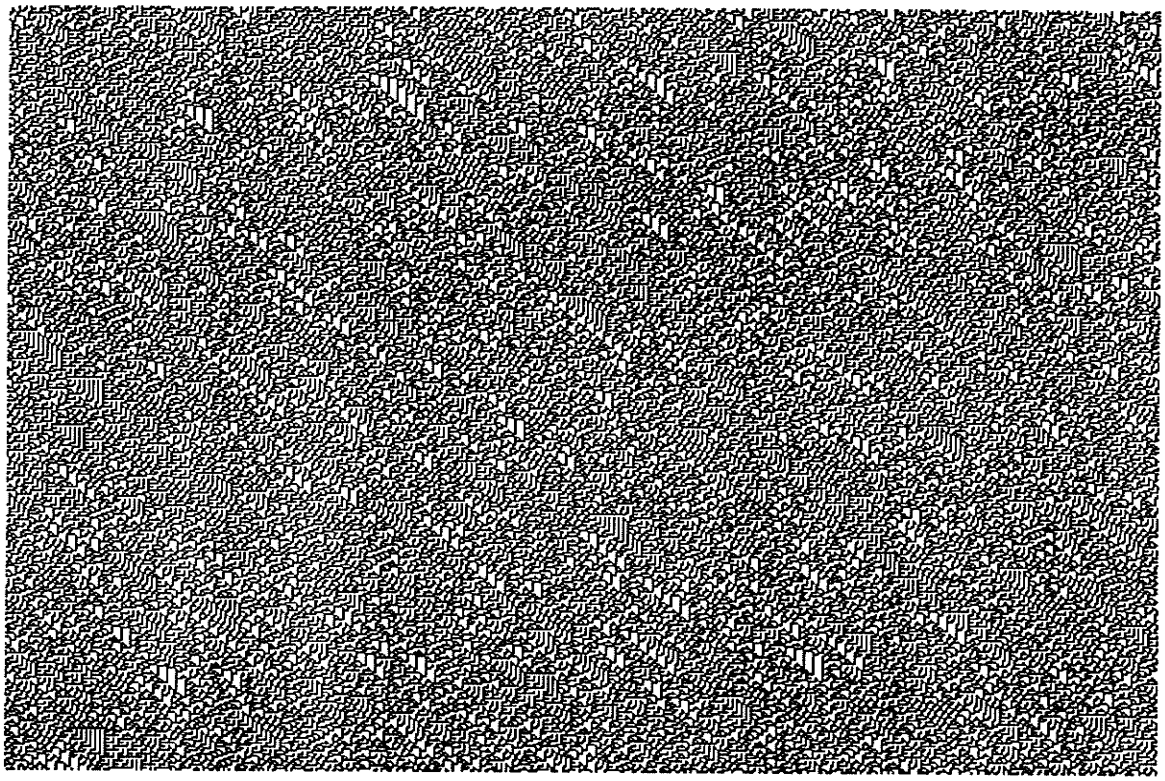


Figure 2.40 : *420 time steps in the state - time diagram of a 500 site rule 30 and 45 hybrid. Cyclic boundary conditions; random initial state.*

150 hybrid with a random initial state and displays its homoplectic properties. As in CA rule 30, we note the presence of triangular patterns in the state - time diagram.

A 30 bit implementation of this particular hybrid uses $1.0 \times 10^6\ \mu m^2$. This compares to $1.1 \times 10^6\ \mu m^2$ for CA rule 30 and $0.46 \times 10^6\ \mu m^2$ for a LFSR of the same register length. The rule 90 and 150 hybrid is slightly smaller than the CA rule 30 and uses only 2.1 times the area of a parallel LFSR. As for CA rule 30, the nearest neighbour wiring leads to a much improved layout over the LFSR. The restriction on size is a minor problem since the constraints do not prohibit a large set of sizes. The null boundary conditions actually provide advantages over CA rule 30 since the first and last sites in the rule 90 and 150 hybrid do not need to be placed in close proximity. The boundary constraints also allow the rule 90 and 150 hybrid to operate at a higher speed since no extended wiring is required.

Unlike CA rule 30, adjacent sites in the rule 90 and 150 hybrid are not correlated in both time and space. This is evident in the auto and cross-correlation data of Fig. 2.38. Thus, cross-correlation in the rule 90 and 150 hybrid is similar to the cross-correlation of the multiplicative congruential PRNG of Fig. 2.19(b). However, the binary sequences produced by sites in the rule 90 and 150 hybrid fail some random number tests because of distribution problems. The fundamental problem with the randomness of the sequence is generation of more ones than zeros in one stage of the binary

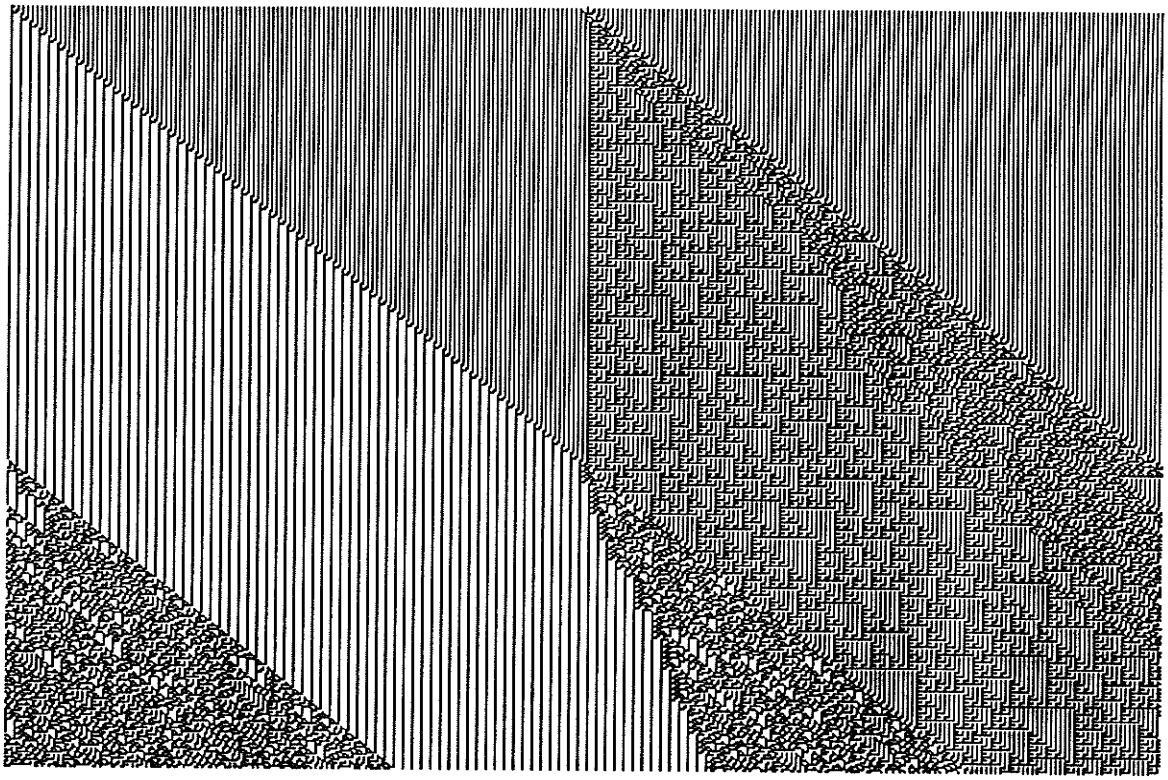


Figure 2.41 : *420 time steps in the state - time diagram of a 501 site rule 30 and 45 hybrid. Cyclic boundary conditions; initialised with a single nonzero site.*

sequence, followed by another stage producing more zeros than ones. This can be problematic in some applications since the distribution is not uniform in small local sequences even though the entire cycle is acceptable. This leads us to speculate that different one-dimensional hybrid cellular automata may exist which have maximal length and a widely acceptable pseudorandom number sequence. However, there is an overwhelming number of possible hybrid rules. Note that hybrid cellular automata are not restricted to only two alternating CA rules but may include other more complicated combinations. Even if we restrict the search to include only one-dimensional nearest neighbour combinations, there are $(2^8)^n$ possibilities for an n bit hybrid cellular automaton.

In the case of CA rule 30 generator time and site spacing were used to remove problems in the generated sequence. The same principle can be applied to this hybrid. In Tables 2.13 and 2.14 the results of the random number tests for various values of site spacing, γ, and time spacing, β, are provided. Notice that a rule 90 and 150 hybrid with a spacing of $\gamma = 0$ possesses little or no random properties because of the distribution problems mentioned previously but, when single spacing site is introduced, the sequences produced pass the tests as well as would be expected of sequences produced by a known good PRNG. This behaviour extends over all the site spaced

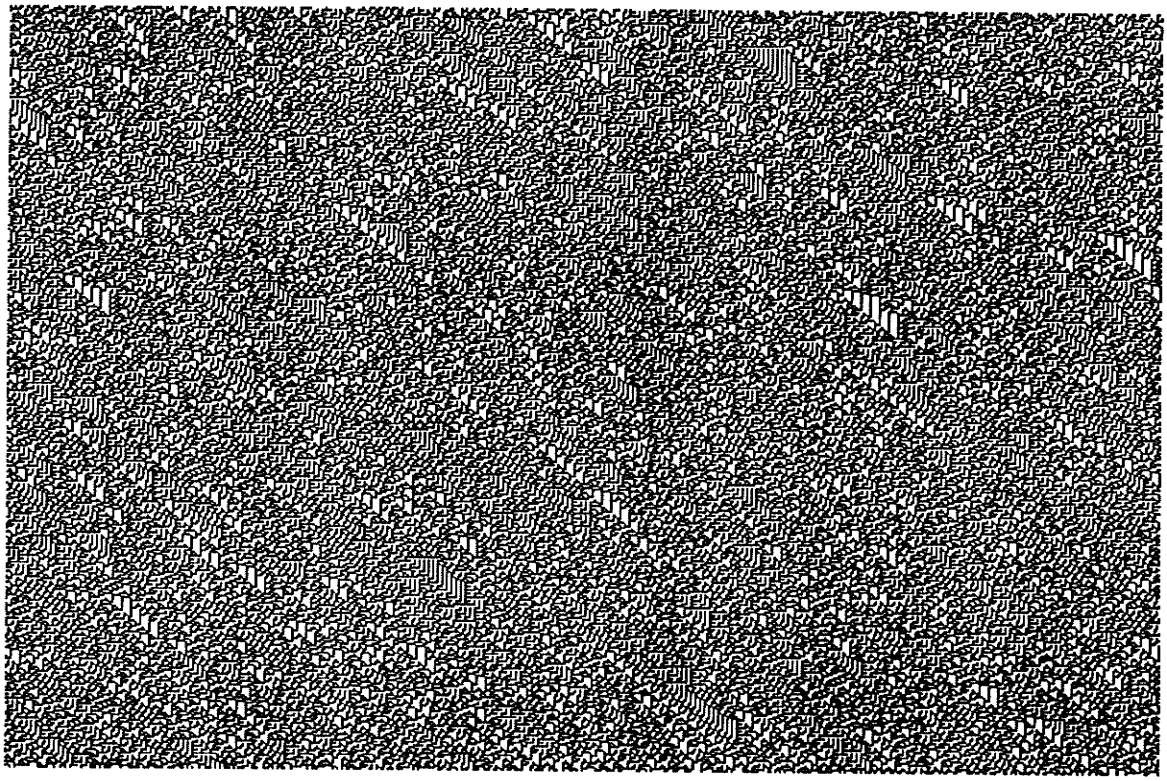


Figure 2.42 : *420 time steps in the state - time diagram of a 501 site rule 30 and 45 hybrid. Cyclic boundary conditions; random initial state.*

rule 90 and 150 hybrid based PRNGs. The only test which the $\gamma = 1$ generator fails is the visual test. However, the other test metrics are well satisfied. Therefore, for the purposes of this work, a site spacing of $\gamma = 1$ will be considered adequate for pseudorandom sequence generation. If this presents unsatisfactory results, then a spacing of $\gamma = 2$ is recommended. Moreover, from the results of Table 2.14 it would seem that time spacing has absolutely no effect on the randomness of the sequence from a rule 90 and 150 hybrid based PRNG. This would appear to be a result of the inadequacy of time spacing to compensate for the bunched distributions of 1's and 0's. A curious result which is left as an open problem.

From Table 2.13 we have established that a rule 90 and 150 hybrid with $\gamma = 1$ will generate sequences which can be considered pseudorandom. This indicates that a rule 90 and 150 hybrid can be used in all the configurations discussed for CA rule 30. However, care should be taken to ensure that the length and boundary restrictions of the hybrid are obeyed. Implementations using the rule 90 and 150 hybrid rather than CA rule 30 will use less area since the area of each cell in the rule 90 and 150 hybrid is smaller. The operational speed of the rule 90 and 150 hybrid is higher because the null boundary conditions ensure that only nearest neighbour communications are used, unlike CA rule 30 where feedback may be necessary. This would seem to lead to the conclusion that the rule 90 and 150 hybrid with $\gamma = 1$ should be used

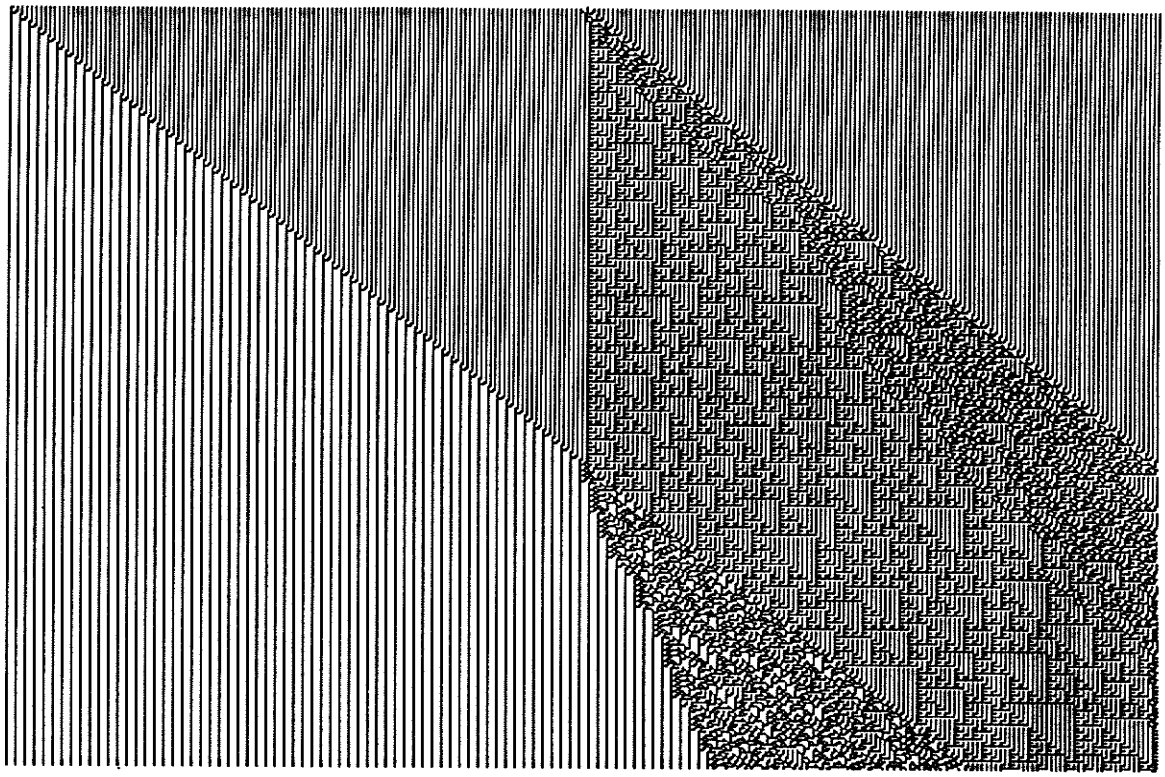


Figure 2.43 : *420 time steps in the state - time diagram of a 500 site rule 30 and 45 hybrid. Null boundary conditions; initialised with a single nonzero site.*

instead of CA rule 30 with $\gamma = 4$ because of its improved area and speed performance. However, care should be exercised in the use of the rule 90 and 150 hybrid based PRNGs since they have not been as carefully studied as the CA rule 30 based PRNGs.

The major advantage of the rule 90 and 150 hybrid is that the sequence produced consists of one large cycle of length $2^n - 1$ and a one-cycle for the zero state. To make a table and figure such as Table 2.8 and Fig. 2.28 for the rule 90 and 150 hybrid is meaningless since the results are already known. However, it should be emphasised that if a long cycle length is a crucial requirement for a particular application then the rule 90 and 150 hybrid with its maximal length cycle may be the implementation of choice.

2.4.6. Another Hybrid

Another hybrid which immediately springs to mind, since its two constituent rules are the two autoplectic CA rules discussed previously, is a combination of CA rules 30 and 45. Here we will add one difference in the construction of the hybrid in that sites 0 and 1 will have the same rule of implementation, all other sites will alternate as before. The reason for this change is to take advantage of CA rule 45's evolution from a zero

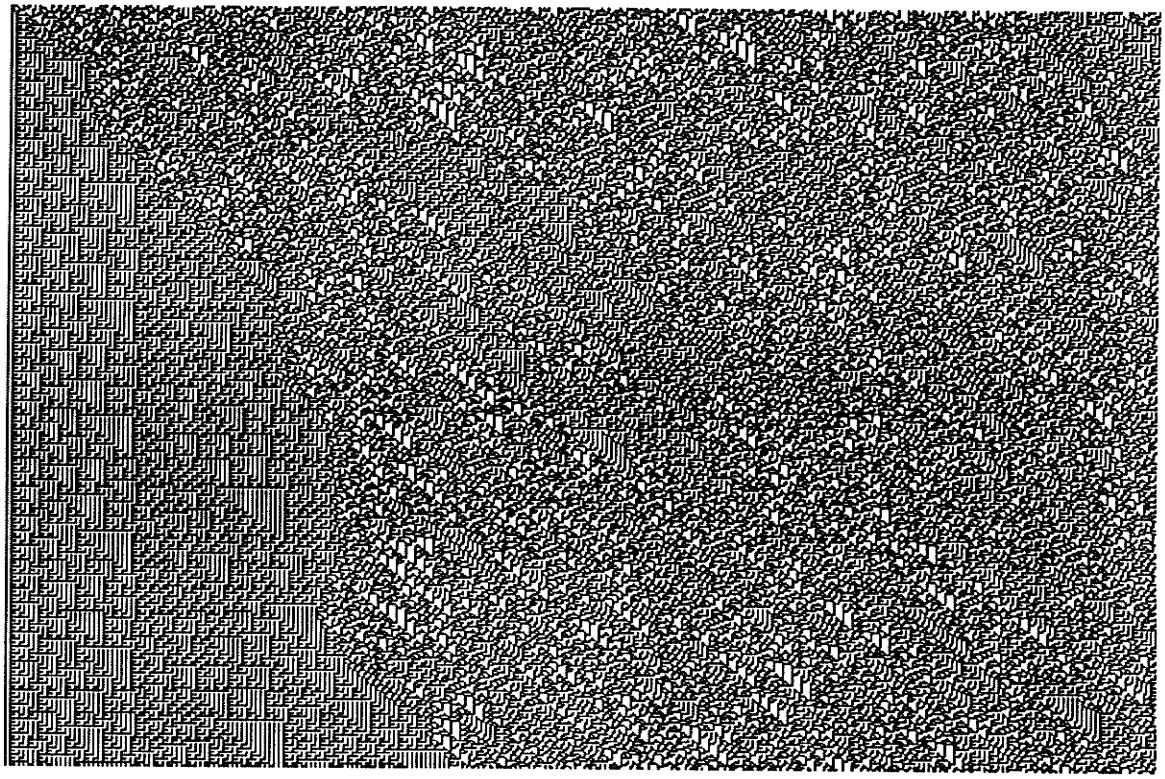


Figure 2.44 : *420 time steps in the state - time diagram of a 500 site rule 30 and 45 hybrid. Null boundary conditions; random initial state.*

state (i.e. we can build a register type PRNG which has a multistate cycle containing the zero state). The resulting state - time diagrams are shown in Figs. 2.39 and 2.40. It should be noted that a complete examination of the rule 30 and 45 hybrid without the twinning of rules at sites 0 and 1 shows no quantifiably different overall behaviour from the rule 30 and 45 hybrid presented here.

The behaviour of this hybrid differs greatly from all previously discussed single initialised site evolution. For the even length rule 30 and 45 hybrid shown in Fig. 2.39 it would appear that the implementation modification has induced a second starting point at the leftmost site. Another difference is that the evolution from the center starting point initially moves left and right but after a few time steps all leftward evolution ceases and subsequent evolution only proceeds to the right. At the induced starting point there is no leftward evolution but rather, what could be termed a moving wall of zeros and ones blocking leftward evolution. The triangular shapes are right triangles rather than the *upside down* equilateral triangles which we have seen thus far. The triangles contain vertical lines of ones and zeros. Finally, the horizontal stripes of CA rule 45 have been replaced by vertical stripes (i.e. a site is staying at value zero or value one until the evolution of states reaches it). For odd size rule 30 and 45 hybrids the behaviour is different from that for even lengths as well as being different from that encountered thus far, as shown in Figs. 2.41 and 2.42. Notice that a secondary

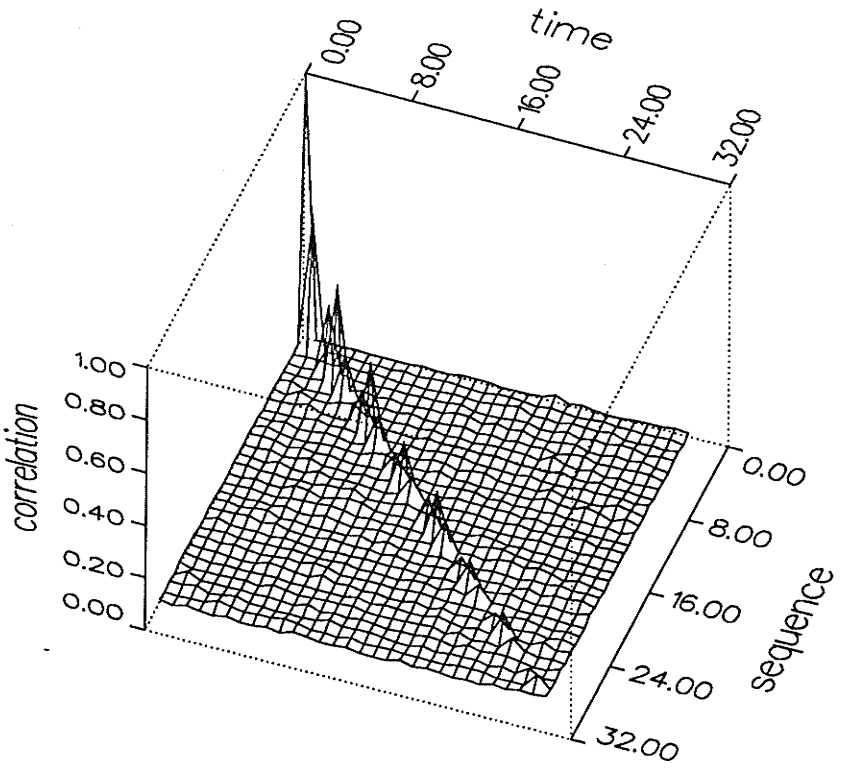


Figure 2.45 : *Cross-correlation of site values in a 30 site rule 30 and 45 hybrid.*

starting point is again induced by the twinned rule at sites 0 and 1. The most significant difference is the vertical height of the diagonal section of vertical stripes. The difference in behaviour between odd and even sized rule 30 and 45 hybrids is not unexpected since in the rule 90 and 150 hybrid we have restrictions on both the length and boundary conditions required to achieve the desired behaviour. For the rule 90 and 150 hybrid the use of linear rules allows the boundary and length conditions to be proofed algebraically [Pries1986] but in the case of the rule 30 and 45 hybrid the chaotic nature of the two rules make such proofs difficult, if not impossible. It can be easily seen that **cyclic** boundary conditions are required for proper behaviour by observing Figs. 2.43 and 2.44 where the use of null boundary conditions induces self-organising behaviour. In order to guarantee some sort of desired behaviour it may be possible to find restrictions on the length experimentally although a number of tests using different lengths, both odd and even, did not reveal fundamentally different properties.

A cause for concern lies in the cross-correlation data of Fig. 2.45. Here the cross-correlation remains large for an even longer period than for CA rule 45 (about 16 sites) and remains distinctly visible across the entire 30 bit hybrid. This would indicate that a large site, or time, spacing would be required in order to remove bit correlation in the pseudorandom word sequences. As with CA rule 45, this suspicion is confirmed in Table 2.15 where the performance of this hybrid using site spacing on the random

Sequence length = 1,000

Test	γ = 0		γ = 1		γ = 2		γ = 3		γ = 4		γ = 5		γ = 6	
mod	10	100	10	100	10	100	10	100	10	100	10	100	10	100
1	100	100	100	100	100	76	100	78	78	78	100	100	100	100
2	75	50	100	100	100	100	100	100	100	100	100	100	100	71
3	75	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100	skip
4	72	skip	100	skip	69	skip	100	skip	55	skip	100	skip	100	skip
5	100	100	78	100	100	100	78	100	100	100	79	100	100	100
6	100	100	100	79	100	100	100	100	100	100	100	79	100	100
7	100	100	79	100	100	100	100	100	100	65	100	100	100	100
8	100	100	79	100	100	100	80	100	100	78	100	100	71	100
9	100	100	78	100	100	100	100	100	100	78	100	100	100	100
10	100	100	79	100	100	100	100	100	80	43	100	100	73	71
11	100	100	100	100	100	100	100	100	80	78	100	100	100	100
12	75	skip	79	skip	85	skip	75	skip	100	skip	79	skip	100	skip
13	72	100	100	76	100	100	100	100	100	100	100	100	100	100
14	100	72	100	100	100	70	100	100	100	100	100	68	100	100
15	skip	100	skip	76	skip	100	skip	100	skip	100	skip	78	skip	100
16	skip	100	skip	79	skip	100	skip	75	skip	55	skip	100	skip	100
17	skip	75	skip	100	skip	39	skip	78	skip	100	skip	100	skip	73
18	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100	skip	71
19	skip	100	skip	100	skip	70	skip	100	skip	77	skip	100	skip	100
20	100	100	100	100	100	100	100	80	100	100	100	100	100	73
21	100	100	76	100	76	100	42	75	100	58	46	54	27	44
22	0.52		0.02		0.29		0.38		0.18		0.02		0.14	
23	Fail		Fail		Fail		Fail		Pass		Pass		Pass	
24	4	4	6	4	3	5	4	4	3	8	3	3	3	4
25	3.31	3.03	3.52	2.90	2.70	3.45	3.25	3.14	2.07	3.90	1.96	2.21	2.29	2.97

Sequence length = 10,000

Test	γ = 0		γ = 1		γ = 2		γ = 3		γ = 4		γ = 5		γ = 6	
mod	10	100	10	100	10	100	10	100	10	100	10	100	10	100
1	59	100	100	100	55	100	100	77	73	100	100	100	47	100
2	100	0	100	100	100	100	100	100	100	100	46	79	100	100
3	100	skip	100	skip	100	skip	100	skip	73	skip	78	skip	79	skip
4	100	skip	100	skip	100	skip	100	skip	100	skip	100	skip	78	skip
5	100	66	73	100	100	100	77	100	73	100	100	100	100	100
6	100	73	100	100	70	100	100	100	100	75	78	79	100	100
7	100	100	100	100	100	100	77	100	100	100	100	100	100	100
8	100	100	71	100	100	100	77	100	100	100	100	67	100	100
9	100	100	100	100	100	100	100	100	100	100	100	67	79	100
10	100	100	100	100	100	100	100	100	100	100	76	67	100	100
11	100	100	100	100	100	100	100	100	100	100	100	100	100	100
12	73	skip	100	skip	100	skip	100	skip	75	skip	100	skip	100	skip
13	100	66	100	100	100	100	100	100	100	100	100	100	100	100
14	73	75	100	100	70	100	100	100	100	100	100	100	100	100
15	skip	100	skip	69	skip	100	skip	100	skip	100	skip	100	skip	100
16	skip	75	skip	71	skip	100	skip	100	skip	74	skip	100	skip	75
17	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100	skip	100
18	skip	100	skip	100	skip	100	skip	77	skip	100	skip	100	skip	57
19	skip	100	skip	71	skip	100	skip	61	skip	100	skip	100	skip	100
20	100	66	100	87	100	100	100	77	100	100	76	78	100	100
21	86	75	71	0	55	100	77	0	51	78	78	100	78	68
22	0.52		0.02		0.29		0.38		0.18		0.02		0.14	
23	Fail		Fail		Fail		Fail		Pass		Pass		Pass	
24	5	6	4	5	6	2	6	7	5	2	4	4	4	2
25	3.09	5.04	2.85	4.02	3.50	2.00	2.92	4.08	2.55	1.73	2.68	2.63	2.39	2.00

Table 2.15: *Random number test results for the rule 30 and 45 hybrid with various site spacing values, γ.*

Sequence length = 1,000

Test	β = 0		β = 1		β = 2		β = 3		β = 4		β = 5		β = 6	
mod	10	100	10	100	10	100	10	100	10	100	10	100	10	100
1	102	100	100	100	98	98	100	100	96	96	98	98	70	98
2	74	51	100	100	98	62	74	100	96	75	98	79	70	98
3	81	skip	79	skip	98	skip	100	skip	96	skip	77	skip	81	skip
4	77	skip	100	skip	62	skip	74	skip	96	skip	98	skip	81	skip
5	102	100	100	100	47	98	100	100	74	100	100	58	100	100
6	102	100	100	100	98	98	81	100	75	96	98	70	98	98
7	102	100	100	100	98	98	100	100	96	96	98	98	98	98
8	102	100	100	100	98	98	100	100	96	96	98	98	98	98
9	102	100	100	100	98	98	100	100	96	68	98	98	98	98
10	100	100	100	100	98	98	100	100	70	68	98	98	98	98
11	100	100	100	100	79	98	100	100	96	96	98	98	98	98
12	79	skip	100	skip	100	skip	100	skip	100	skip	51	skip	100	skip
13	77	100	100	100	98	98	100	100	96	96	98	98	98	98
14	100	77	100	100	81	98	100	100	96	96	98	98	98	98
15	skip	100	skip	81	skip	98	skip	100	skip	96	skip	98	skip	98
16	skip	100	skip	100	skip	98	skip	100	skip	68	skip	98	skip	98
17	skip	72	skip	81	skip	98	skip	100	skip	96	skip	98	skip	98
18	skip	100	skip	100	skip	98	skip	81	skip	96	skip	98	skip	98
19	skip	100	skip	68	skip	98	skip	74	skip	47	skip	98	skip	98
20	100	100	100	60	98	98	74	100	96	96	98	98	55	72
21	100	100	100	79	98	83	100	100	55	75	77	51	98	72
22	0.52		0.02		0.29		0.38		0.18		0.02		0.14	
23	Fail		Fail		Fail		Fail		Pass		Pass		Pass	
24	4	4	3	5	16	18	5	3	15	18	15	19	15	18
25	3.17	3.00	2.21	3.30	3.51	2.85	2.98	2.45	2.68	3.40	2.17	2.68	2.60	1.85

Sequence length = 10,000

Test	β = 0		β = 1		β = 2		β = 3		β = 4		β = 5		β = 6	
mod	10	100	10	100	10	100	10	100	10	100	10	100	10	100
1	50	100	73	74	100	100	100	100	100	100	76	100	85	85
2	100	0	73	73	78	100	79	100	51	81	74	100	100	100
3	100	skip	100	skip	100	skip	100	skip	70	skip	100	skip	100	skip
4	100	skip	100	skip	78	skip	100	skip	100	skip	81	skip	85	skip
5	100	77	100	79	72	72	100	100	70	100	100	100	49	75
6	100	66	100	74	100	100	100	100	100	100	100	81	100	100
7	100	100	100	100	100	78	100	100	100	100	100	76	100	100
8	100	100	100	100	100	100	100	100	100	100	100	100	100	100
9	100	100	100	100	100	100	79	79	100	100	100	76	100	100
10	100	100	100	100	100	100	100	100	100	100	81	100	100	100
11	100	100	100	100	100	100	100	76	70	100	100	100	100	100
12	66	skip	73	skip	100	skip	100	skip	61	skip	100	skip	100	skip
13	100	77	100	100	100	100	100	100	100	81	100	100	100	100
14	66	73	100	100	100	100	100	71	100	80	69	100	66	100
15	skip	100	skip	73	skip	77	skip	76	skip	100	skip	100	skip	100
16	skip	73	skip	74	skip	78	skip	45	skip	100	skip	100	skip	74
17	skip	100	skip	73	skip	77	skip	100	skip	100	skip	100	skip	100
18	skip	100	skip	100	skip	100	skip	74	skip	100	skip	100	skip	100
19	skip	100	skip	52	skip	73	skip	74	skip	100	skip	69	skip	100
20	100	77	73	79	100	72	74	100	70	100	100	100	85	74
21	84	73	53	47	100	51	47	55	100	50	57	55	74	75
22	0.52		0.02		0.29		0.38		0.18		0.02		0.14	
23	Fail		Fail		Fail		Fail		Pass		Pass		Pass	
24	5	6	5	6	4	5	4	5	6	3	3	2	4	3
25	3.34	4.84	2.55	4.02	2.72	4.22	3.21	4.50	3.08	2.08	2.62	2.43	2.56	2.17

Table 2.16: *Random number test results for the rule 30 and 45 hybrid with various time spacing values, β.*

N	Cycles	Frac. longest
4	1x7, 1x1	0.94
5	1x4	1.00
6	1x14	1.00
7	1x13, 1x6, 1x1	0.94
8	1x35, 1x30, 1x8, 1x1	0.34
9	1x15	1.00
10	1x335, 1x45, 1x16, 1x13	0.93
11	1x27, 1x22, 1x14, 1x1	0.97
12	1x311, 1x111, 1x101, 3x12, 1x5, 1x1	0.53
13	1x281, 1x263, 1x231, 1x15	0.71
14	1x543, 1x100, 1x61, 1x32, 1x16, 1x5	0.68
15	1x1211, 1x993, 1x15, 1x1	0.77
16	1x4962, 1x1090, 1x1060, 1x10, 1x8, 1x1	0.67
17	1x6183, 1x1147, 1x98, 1x19, 1x10	0.56
18	1x4454, 1x4174, 1x1318, 1x644, 1x90, 1x56, 1x14, 1x7, 1x5	0.08
19	1x4834, 1x4795, 1x2156, 1x2042, 1x1755, 1x544, 1x235, 1x140, 1x50, 1x22, 1x18, 1x7, 1x6, 1x1	0.07
20	1x11413, 1x3309, 1x1723, 1x1331, 1x1246, 1x270, 1x57, 1x48, 1x5, 1x1	0.03

Table 2.17: *Cycles lengths for various size rule 30 and 45 hybrids and the fraction of all states leading to the longest cycle.*

number tests is given. Table 2.15 shows that despite the *good randomness* properties at small site spacings the randomness of the rule 30 and 45 hybrid based PRNG does not rapidly improve with site spacing. Unlike the rule 90 and 150 hybrid, the use of time spacing yields improvements in the randomness of the pseudorandom sequences as evidenced in Table 2.16, but as with site spacing, the improvement is very slow. An important point to note is that when no site, or time, spacing is used the performance of the rule 30 and 45 hybrid based PRNG is better than that of the CA rule 30 or 45 based PRNGs.

If we consider the cycle lengths of this hybrid, given in Table 2.17 and the plot of Fig. 2.46, a similar difficulty as with the CA rule 45 arises (i.e. no one dominant long cycle and a lower percentage of states leading to the longest cycle). However, the length of the longest cycle is now comparable to that produced by CA rule 30, so the cycle length properties of this hybrid consist of the poor qualities of its two constituent rules. This compares to the rule 90 and 150 hybrid where one maximal length cycle is used. However, the rule 30 and 45 hybrid possesses one very unique property over all the cellular automata discussed thus far in that the zero state is not a one or two cycle. For example, consider Fig. 2.47 where the rule 30 and 45 hybrid was initialised with a zero state. Notice that what was previously termed the induced secondary starting point now acts as a primary starting point and eventually leads to random behaviour even though the evolution is only to the right.

In comparing the two hybrids we see that nearly equivalent randomness can be shown on the basis of the random number tests. However, the bit sequence correlation is much higher in the rule 30 and 45 hybrid in addition to possessing smaller

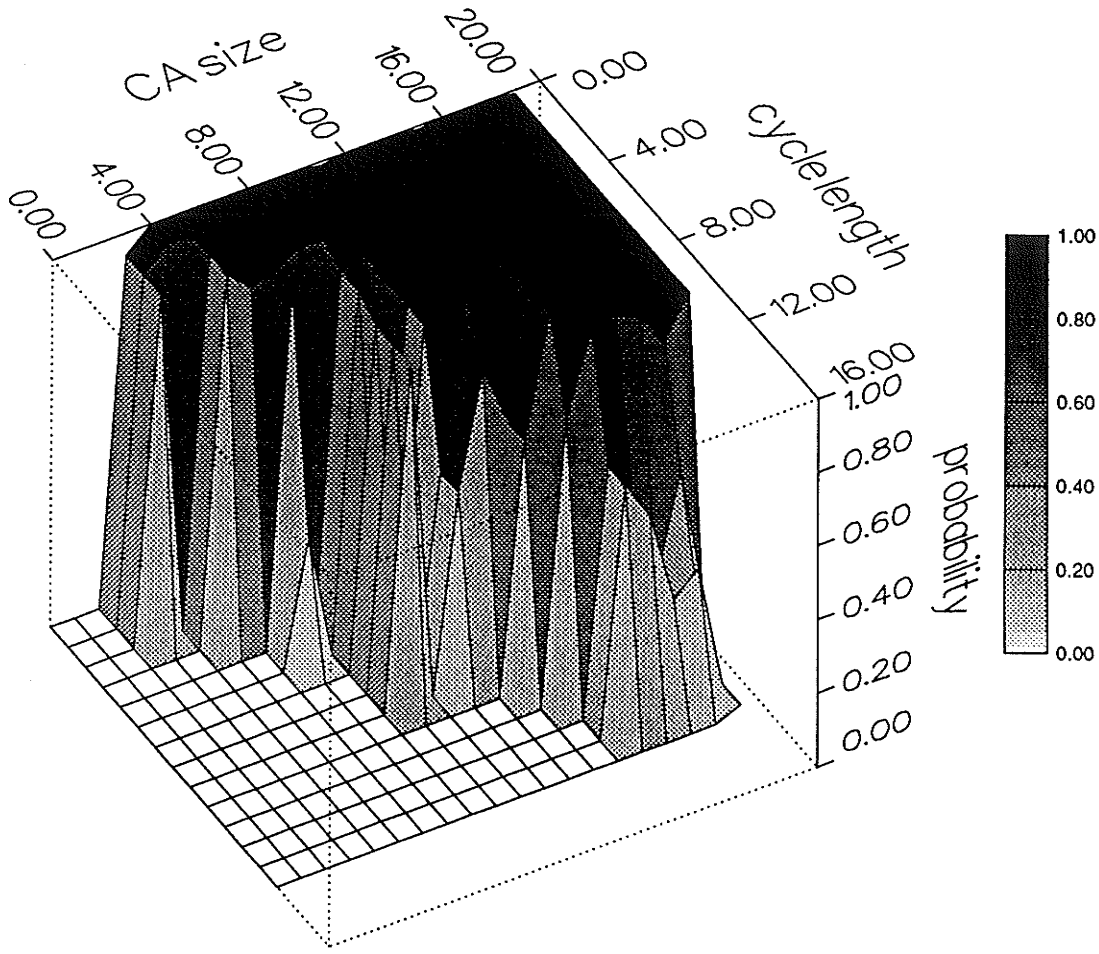


Figure 2.46 : *Cycle length versus probability of entering a cycle of that length for various rule 30 and 45 hybrids with cyclic boundary conditions. See appendix B for complete tables.*

cycles and slightly larger area for implementation. Therefore, the rule 90 and 150 hybrid is presently considered to be the hybrid of choice except in cases where the ability of the rule 30 and 45 hybrid to include the zero state in a long cycle outweighs these considerations.

2.4.7. More Complicated Cellular Automata

In the previous sections we have considered only one-dimensional elementary cellular automata and elementary hybrids. The purpose of this section is to give a brief introduction to more complicated cellular automata so that the reader may appreciate their potential for applications other than those discussed in this work.

The first added complication will be to consider simple linear arrays, as before, except that the value emanating from each site is no longer restricted to modulo 2 (i.e. binary) values but rather, can have k possible values. In this case the number of

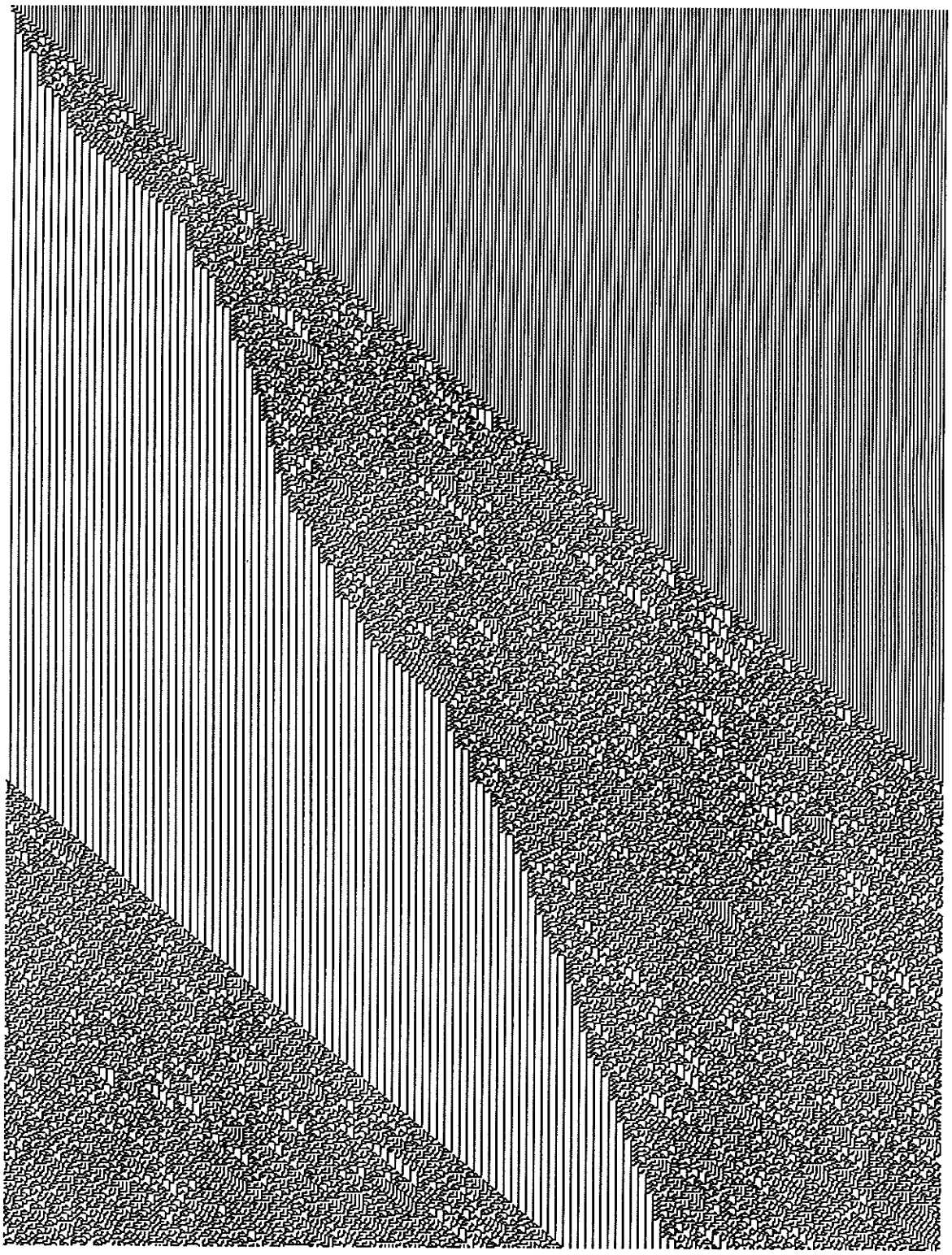


Figure 2.47 : *800 time steps in the state - time diagram of a 500 site rule 30 and 45 hybrid. Cyclic boundary conditions;* **zero** initial state.

possible CA rules, and thus the number of different cellular automata of type modulo k, increases to

$$k^{k^3} \tag{2.40}$$

for nearest neighbour cellular automata. Obviously this number increases very rapidly with k. For $k = 3$ there are $3^{3^3} \approx 10^9$ possible rules and for $k = 4$, $4^{4^4} \approx 10^{154}$. This large number of possibilities indicates the potential of cellular automata and also one of its main weaknesses. That is, there are limitless possibilities for CA implementations but, at the same time, it requires more than the *life time of the universe* to investigate all the possibilities. In this work modulo $k > 2$ cellular automata are not considered but the reader should be aware of their existence. It is possible that this form of CA may have applications in some of the computational physics problems considered in Chapter 3.

As yet, we have only considered cellular automata where each site is solely a function of itself and the left and right neighbours, i.e.

$$a_i(t+1) = \phi\left[\, a_{i-1}(t)\,,\, a_i(t)\,,\, a_{i+1}(t) \,\right] \quad . \tag{2.41}$$

We can define a more general one-dimensional neighbourhood as

$$a_i(t+1) = \phi\left[a_{i-r}(t),\, a_{i-r-1}(t),\, \cdots\, a_i(t)\, \cdots\, ,\, a_{i+r}(t)\right] \quad . \tag{2.42}$$

Therefore, we now consider r neighbours to the left and right or $2r + 1$ neighbours in all. Now there are

$$k^{k^{2r+1}} \tag{2.43}$$

possible rules for modulo k cellular automata. For $k = 2$ and a neighbourhood of $r = 2$ we have $2^{2^5} \approx 4 \times 10^9$ possible CA rules, if $k = 3$ this increases to $3^{3^5} \approx 8 \times 10^{115}$. As before, the number of possible cellular automata becomes effectively innumerable and so, impossible to thoroughly investigate. A problem that occurs when the neighbourhood is increased is greater wiring overhead. This would probably not be a problem for $r = 2$ neighbourhoods but for larger neighbourhoods the area penalty could start to become significant. It is possible that the increased inter-dependence of this type of CA could lead to better pseudorandom number generation since one would expect bit sequence correlations to be reduced by the larger neighbour set. Some preliminary investigations were carried out by the author but no distinctly better PRNGs were found. However, a thorough search was not carried out and it is certainly possible that a noticeably better CA-based PRNGs over those with a neighbourhood of $r = 1$ exist.

A final complication concerns the addition of more than one dimension. For example, a two-dimensional grid, such as that of Fig. 2.1, could be easily fashioned. The general problem of two-dimensional cellular automata has not been as thoroughly studied as that of one-dimensional cellular automata although some particular two-dimensional cellular automata, such as the so called game of *Life* [Berlekamp1982], [Gardner1971] have had their evolutions very thoroughly studied. There are two main

neighbourhoods for two-dimensional cellular automata. One consists of a *five-neighbour square* where top, bottom, left, and right neighbours are used. The other consists of a *nine-neighbour square* where the top left and right and bottom left and right squares are added to the previous neighbourhood [Packard1985b]. These two neighbourhoods are often referred to as the von Neumann and Moore neighbourhoods, respectively. It is also possible to define hexagonal and triangular lattices as the neighbourhood but these have been even less extensively investigated than the square von Neumann and Moore neighbourhoods. The number of possible two-dimensional cellular automata is, in general,

$$k^{k^r} \tag{2.44}$$

where r is the number of sites in the neighbourhood (remember to include a site as its own neighbour) and k is the modulus. It can be seen that even for the small von Neumann neighbourhood the number of possible CA rules becomes prohibitively large for a thorough investigation ($\approx 4 \times 10^9$). As for the one-dimensional cellular automata, no direct mathematical techniques exist and so, two-dimensional cellular automata can only be studied by observing their evolution over time. This can take a prohibitively long time. In [Packard1985b] a special purpose CA simulation engine [Toffoli1984], [Toffoli1987] is used to make an initial study of two-dimensional cellular automata. Packard's study of two-dimensional cellular automata has shown that there is a correspondence between the four global classes of behaviour in the one-dimensional case and global behaviour in the two-dimensional case. This means that the results of [Packard1985b] indicate that there exists in two-dimensional cellular automata the equivalent of one-dimensional class 3 behaviour. These are important results, for if such two-dimensional cellular automata exist, then it may be possible to find a two-dimensional cellular autmaton which could be imbedded in the mesh layouts under consideration. This could avoid some of the problems found in imbedding one-dimensional cellular automata into the mesh architecture. However, much remains to be investigated before such two-dimensional cellular automata could be proposed.

2.5. SOME CONCLUSIONS

The random number test results of some of the various generators discussed in this chapter are summarised in Table 2.18. Here only CA rule 30 and the rule 90 and 150 hybrid are considered since the behaviour of the other CA-based PRNGs have similar or worse randomness characteristics. As expected, the algorithmic generators produce sequences which pass the given random number tests and yield an evenly weighted average failure of about 1.1 to 1.5 and a worst case performance of 4.0 to 7.0. The serial-in/parallel-out LFSR-based PRNG produces sequences which provide comparable pseudorandomness. As stated previously, the parallel LFSR produces sequences which are woefully inadequate as pseudorandom sequences. What is surprising is the performance of the CA rule 30 based PRNG with $\gamma = 0$. This generator, while certainly not as random as the algorithmic PRNGs, provides sequences

which do not consistently perform poorly on any one given test. This indicates that the sequences may be useful in some applications where only a fairly random sequence is required rather than a completely random sequence. Remember that the rule 30 and 45 hybrid yields better performance if no site spacing is used. CA rule 30 with $\gamma = 4$ and the rule 90 and 150 hybrid with $\gamma = 1$ give comparable results to the algorithmic PRNGs and so, we can consider the sequences produced by these generators to be pseudorandom. Previously it was stated that from the tables we are 75% confident that less than 10% of the sequences produced by the PRNGs under consideration here produce nonpseudorandom sequences. To further increase this confidence it is essential that we run a large number of sequences through the random number tests. This was done for CA rule 30 with $\gamma = 4$ and the rule 90 and 150 hybrid with $\gamma = 1$ so that, using the analysis of Appendix A, we are 97.5% confident that less than 1% of the sequences produced will not be pseudorandom.

Another measure to use in comparing PRNGs is the area used per bit of pseudorandom number as given in Table 2.19. Notice that of the *good* generators the smallest area per bit measure is given by the $\gamma = 1$ rule 90 and 150 hybrid. This table indicates that this is the optimal structure. However, as mentioned previously, care should be exercised in its use since structures of this type have only recently been proposed and are still being fully investigated. The next smallest area per bit measure of a *good* CA-based generator is achieved by CA rule 30 with $\gamma = 4$. The only other pseudorandom number generation techniques which provide comparable area per bit measures to the cellular automata based generators are those using the LFSR. We can immediately rule out the parallel LFSR technique with no wait states since it produces decidedly nonrandom sequences. However, the quality of randomness from both the serial-in/parallel-out method and the parallel LFSR with $\beta = n$ is similar to that of the CA-based PRNGs. The serial-in/parallel-out method consumes only slightly more area than the rule 90 and 150 hybrid with $\gamma = 1$ but it does not possess the same layout advantages as the rule 90 and 150 hybrid. This is because the LFSR cannot be laid out in a regular fashion since the feedback taps are register length dependent. This causes problems as the length of the LFSR is changed to reflect changes in the architecture. In addition, the layout must provide for the irregular placement of the *exclusive-or* gates in the LFSR. Finally, the LFSR requires global communication since a feedback path exists between the beginning and the end of the LFSR. Therefore, the rule 90 and 150 hybrid with $\gamma = 1$ and CA rule 30 with $\gamma = 4$ based PRNGs should be preferred over the serial-in/parallel-out LFSR based PRNG.

The parallel LFSR with $\beta = n$ uses about a quarter of the area of the rule 90 and 150 hybrid with $\gamma = 1$, but possesses the layout problems of the LFSR. More importantly, pseudorandom numbers appear only every n clock cycles giving them a very poor time performance in comparison to the CA-based PRNGs. For example, if the word size of the pseudorandom numbers is 30 bits then the parallel LFSR with $\beta = n$ will use a factor of 30 more time to generate each pseudorandom number. Therefore, the AT measure of the parallel LFSR with $\beta = n$ is much worse than the CA-based generators if the word size is appreciable. It can be shown that equivalent AT metrics

Sequence length 1,000

Test	Box Muller		Mult Cong		Additive Feedback		Serial-in Parallel-out		Parallel LFSR		Rule 30 CA γ = 0		Rule 30 CA γ = 4		Hybrid CA γ = 1	
mod	10	100	10	100	10	100	256	8	10	100	10	100	10	100	10	100
1	93	97	91	96	93	91	100	76	78	100	100	100	100	100	57	76
2	89	89	93	94	94	86	100	76	0	0	100	72	100	71	100	100
3	89	skip	93	skip	94	skip	100	skip	0	skip	74	skip	100	skip	100	skip
4	86	skip	90	skip	88	skip	100	skip	0	skip	100	skip	81	skip	81	skip
5	95	99	97	93	93	95	70	100	100	100	72	100	100	100	57	100
6	96	93	94	96	94	95	100	75	22	45	100	100	100	100	100	100
7	93	97	93	96	95	98	100	100	33	78	100	100	38	100	100	100
8	96	98	95	100	91	96	79	100	67	22	100	100	100	100	100	100
9	94	98	94	97	99	96	100	100	100	78	100	100	100	100	100	100
10	93	97	94	93	97	96	75	100	67	22	72	100	100	100	100	77
11	97	99	98	96	95	99	100	100	67	22	100	100	100	100	100	100
12	92	skip	92	skip	93	skip	100	skip	77	skip	100	skip	100	skip	81	skip
13	98	99	98	99	98	97	100	100	100	45	100	100	100	100	100	100
14	95	94	94	92	97	96	100	100	78	33	100	100	100	100	100	100
15	skip	89	skip	92	skip	90	skip	70	skip	22	skip	100	skip	100	skip	100
16	skip	88	skip	90	skip	93	skip	79	skip	22	skip	100	skip	100	skip	100
17	skip	91	skip	81	skip	98	skip	100	skip	0	skip	100	skip	100	skip	100
18	skip	94	skip	94	skip	91	skip	100	skip	67	skip	100	skip	100	skip	100
19	skip	94	skip	91	skip	91	skip	100	skip	55	skip	100	skip	100	skip	100
20	94	89	89	93	88	81	79	76	22	55	100	72	100	100	23	100
21	75	66	78	65	70	70	70	30	0	0	26	74	81	81	53	100
22	0.08		0.09		0.12		0.05		1.0		0.50		0.07		0.05	
23	Pass		Pass		Pass		Pass		Fail		Fail		Pass		Fail	
24	5	6	7	5	6	5	2	4	11	16	5	4	2	1	6	2
25	1.25	1.29	1.17	1.42	1.21	1.41	1.27	2.18	9.89	12.3	3.56	2.82	1.00	0.48	3.48	1.47

Sequence length 10,000

Test	Box Muller		Mult Cong		Additive Feedback		Serial-in Parallel-out		Parallel LFSR		Rule 30 CA γ = 0		Rule 30 CA γ = 4		Hybrid CA γ = 1	
mod	10	100	10	100	10	100	256	8	10	100	10	100	10	100	10	100
1	96	93	96	95	90	91	100	75	100	100	81	81	100	80	100	100
2	97	85	89	93	93	95	100	100	0	0	81	0	50	100	100	100
3	94	skip	92	skip	90	skip	100	skip	0	skip	48	skip	100	skip	100	skip
4	91	skip	97	skip	89	skip	100	skip	0	skip	100	skip	100	skip	100	skip
5	96	93	97	93	93	94	100	100	100	100	100	100	100	100	100	100
6	95	97	96	97	96	96	100	74	0	0	100	100	100	100	100	100
7	91	99	96	98	91	98	100	100	0	0	100	100	80	100	100	100
8	93	96	98	97	92	92	100	74	0	0	71	100	100	70	100	82
9	91	96	96	98	93	100	100	100	0	0	100	100	100	100	100	100
10	99	92	93	95	95	97	100	100	0	0	81	81	100	100	100	100
11	91	98	96	98	97	98	100	100	0	0	100	79	100	100	100	100
12	93	skip	90	skip	96	skip	100	skip	100	skip	79	skip	100	skip	76	skip
13	99	94	98	99	93	95	78	74	100	100	100	100	80	100	100	100
14	95	95	96	95	91	93	100	100	0	0	100	71	100	100	100	100
15	skip	87	skip	86	skip	91	skip	74	skip	0	skip	81	skip	100	skip	100
16	skip	90	skip	90	skip	95	skip	100	skip	0	skip	100	skip	100	skip	100
17	skip	87	skip	88	skip	88	skip	73	skip	0	skip	100	skip	100	skip	100
18	skip	89	skip	88	skip	89	skip	75	skip	0	skip	100	skip	80	skip	100
19	skip	88	skip	92	skip	88	skip	100	skip	0	skip	100	skip	75	skip	100
20	94	89	93	91	92	88	100	100	0	0	79	50	100	80	100	76
21	67	74	67	66	63	68	73	78	0	0	0	60	25	80	82	72
22	0.03		0.03		0.03		0.05		1.0		0.50		0.07		0.05	
23	Pass		Pass		Pass		Pass		Fail		Fail		Pass		Fail	
24	4	5	5	4	5	7	1	4	14	16	6	7	3	4	2	2
25	1.18	1.58	1.10	1.41	1.46	1.44	0.49	2.03	14.0	16.0	4.80	4.97	1.65	1.35	1.42	1.70

Table 2.18: *Various pseudorandom number generators and their test results*

Generator	Area per bit[a]	Quality of Randomness
Box Muller	N/A	Good[d]
Multiplicative Congruential	745	Good[d]
Additive Feedback	436	Good[d]
Serial-in Parallel-out LFSR	75	Good[d]
Parallel LFSR	15	Poor[b]
CA Rule 30 $\gamma = 0$	36	Fair[c]
CA Rule 30 $\gamma = 4$	180	Good[d]
CA Rule 45 $\gamma = 0$	43	Fair[c]
Rule 90 and 150 Hybrid $\gamma = 0$	33	Fair[c]
Rule 90 and 150 Hybrid $\gamma = 1$	66	Good[d]
Rule 30 and 45 Hybrid $\gamma = 0$	40	Fair[c]

[a] in $10^3\ \mu m^2$.
[b] Totally inadequate randomness properties due to correlation.
[c] Good for use in some applications.
[d] Good for use in all applications.

Table 2.19: *Area used per PRNG bit by various pseudorandom number generators.*

for the parallel LFSR with $\beta = n$, and for the rule 90 and 150 hybrid with $\gamma = 1$, are achieved at a word size of 5 bits, and for CA rule 30 with $\gamma = 4$, at a word size of 13 bits. The final choice of which PRNG to use is dependent on the possible size of the PRNG and the importance of the time delay between pseudorandom numbers. For the fine-grained processor architectures proposed in this work it would appear that CA-based PRNGs are preferable because of layout, time delay, and quality of random numbers.

A final consideration lies in the cycle length of the sequences produced by the PRNG. Algorithmic generators, such as the R250 additive feedback PRNG, have very large cycle lengths (2^{250}) since it is assumed that necessary resources, such as data memory, are available. However, we have shown that the overhead of supplying these resources is too great for use in the architectures considered here. Therefore, our choices are again reduced to considering only the CA and LFSR-based PRNGs. If we consider CA rule 30 then the cycle lengths of the LFSR-based generators is much greater (2^N versus $2^{.6N}$) which means that a longer CA rule 30 based PRNG is required (1/ 0.6 = 1.6 times longer) to achieve similar cycle length. This is a major deterrent to using CA rule 30 based PRNGs. However, the cycle length of CA rule 45 and the rule 90 and 150 hybrid is similar to that of the LFSR. Note that for applications which require word sizes of ≥ 30 bits and good quality randomness (i.e. site spacing), if a CA rule 30 based PRNG is used the site spacing requirement will force us to use a cellular automaton which is at least 120 bits long and so we will have a cycle length of 2^{72} which should be long enough for most applications. Therefore, cycle length is

not a great concern when using CA-based PRNGs since sufficient cycle length for most applications can be easily obtained.

Chapter 3
Parallel Architectures for Statistical Mechanics

3.1. INTRODUCTION

The emergence of special-purpose computers which exploit the properties of very-large-scale integration (VLSI) as an implementation medium has been most obvious in digital signal processing applications [Denyer1985], [Kung1985]. Among the most successful approaches are the so-called systolic arrays, which recognise the importance of local communications and high degrees of concurrency [Kung1980]. Systolic arrays have primarily been employed in the implementation of deterministic algorithms which take the form of matrix-vector or matrix-matrix multiplications (1-D or 2-D systolic arrays respectively). These arrays are a special case of cellular automata [Wolfram1983], [Burks1970], [Codd1968] in which the nodes are of intermediate complexity between fine-grained cellular automata and microprocessor arrays [Seitz1984].

In this chapter, we will also employ cellular automata arrays, but the focus is upon nondeterministic algorithms, specifically those commonly used in statistical mechanics. One of the important ingredients in this approach is the efficient implementation of distributed pseudorandom number generation over the array. In this approach we will use the concepts developed in Chapter 2 based on the recent discovery that effectively random behaviour may be induced in elementary or primitive one-dimensional (1-D) cellular automata arrays even though the local logical rules are deterministic [Wolfram1984a], [Wolfram1986a].

In this chapter novel architectures for two common statistical mechanical models, (percolation and Ising) will be proposed. These models can be adapted to solve a number of parallel nondeterministic problems, employing algorithms such as parallel Monte Carlo simulations [Wallqvist1987], simulated annealing, both serial [Kirkpatrick1983] [Vecchi1983] and parallel [Darema1987a] [Darema1987b], and phase transition problems [Stanley1971].

3.2. THE PERCOLATION MODEL

The percolation model was first formally defined and studied in [Broadbent1957], although it had been previously studied for many years under different names. The subject was first reviewed by [Frisch1963]. Since then a number of review articles

have appeared studying various aspects of the percolation model. [Shante1971] studied the general percolation problem and related percolation to several different physical processes. [Essam1972] studied the combinatorial aspects of the problem as well as the behaviour near the percolation threshold and [Kirkpatrick1973] studied the relationship between percolation theory and electrical transport in random resistor networks.

The fundamental ideas behind percolation theory are actually quite simple. Here we will use a typical explanation such as that in [Thouless1978]. Consider an infinite solid with a random distribution of equal sized *holes*. Let the presence of a hole in the solid be governed by a probability, p. Therefore, we expect the density of holes to be equal to p. For small values of p most holes will be isolated, with only a few holes combining to form a *cluster* of holes. As p is increased the number of clusters and their size grows. This continues until at some value of p an *infinite* cluster is formed. At this value of p, usually termed the critical probability or percolation threshold, p_c, a path is suddenly formed between opposite faces of the solid. It is now possible for a liquid to *percolate* throughout the solid from one side to another. If p is still further increased the solid will eventually no longer be connected to itself and will fall into pieces. An example is shown in Fig. 3.1 for a square two-dimensional lattice. In Fig. 3.1(a) we see that for $p \ll p_c$ the holes are isolated with only a few small clusters; for $p \lesssim p_c$ (Fig. 3.1(b)) the number and size of the clusters has grown but still no path exists from top to bottom or left to right. However, when $p \gtrsim p_c$ (Fig. 3.1(c)) a path exists from top to bottom suggesting that, in an infinite lattice, an infinite cluster has been formed. Note that some smaller finite clusters still exist. For $p \gg p_c$ (Fig. 3.1(d)) the material consists almost solely of holes and has been broken up into a number of smaller pieces.

Formally, the critical probability is defined as the largest value of p for which the probability of forming an infinite cluster is zero. This only applies to an infinite lattice. Above we considered *site* percolation since p governed the existence of a hole in the lattice. It is also possible to consider *bond* percolation where we assign p to govern the existence of a bond between various holes, or sites, in the lattice. Another common version of the percolation model is *site-bond* percolation where p and q govern the probability of a site or bond respectively. The value of p_c is dependent on the lattice type and dimension. There is also a weak dependence of p_c', the site probability for a percolation probability of 1/2 on a finite lattice, upon the lattice size. In Fig. 3.2 we see a plot of the probability of percolation versus site probability for various size square lattices. A table showing values of p_c for various infinite, or at least very large, lattices for both site and bond percolation is given in Table 3.1.

A natural question which might arise at this point is, that while there is a certain abstract beauty of the percolation model, what is the practical significance of this model. In fact, percolation is a very important model since it is applicable to a wide variety of physical phenomena. The most important property of the percolation model lies in the sharp phase transition from finite to infinite cluster which occurs with

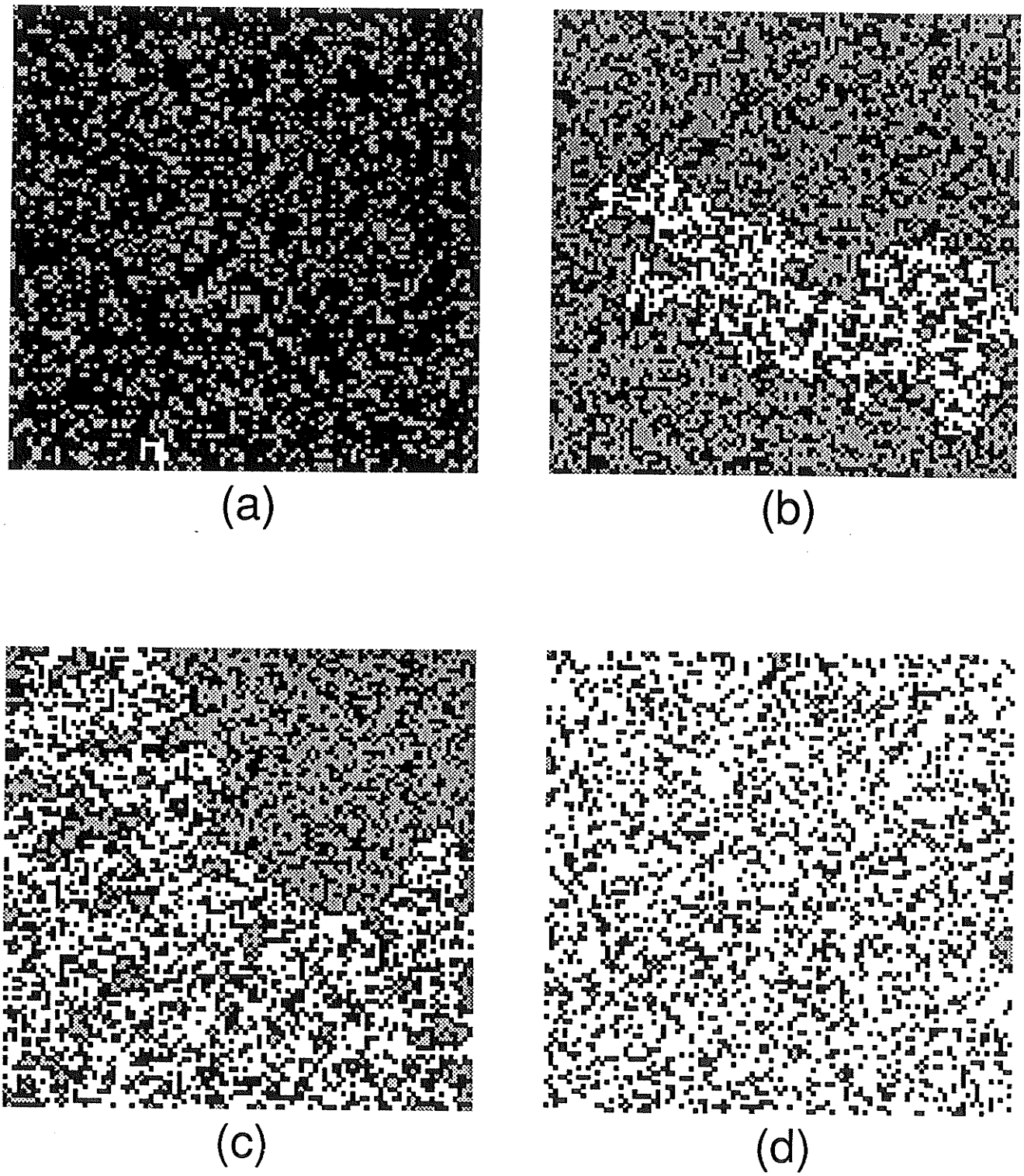


Figure 3.1 : *Two-dimensional illustration of percolation on a 100 x 100 square lattice, p_c = 0.5928. Solid shown in black, holes shown in grey, the largest cluster of holes is highlighted in white. (a) p = 0.25. (b) p = 0.585. (c) p = 0.60. (d) p = 0.75.*

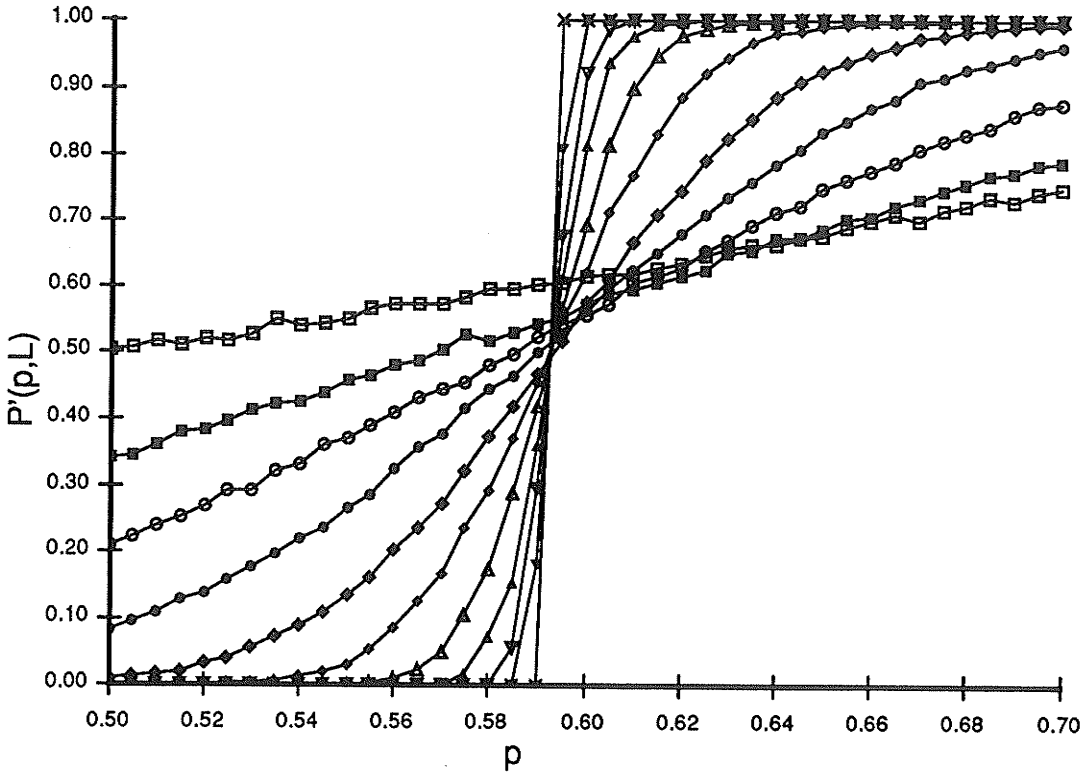


Figure 3.2 : *Probability of percolation, $P'(p,L)$ versus site probability, p, for various lattice sizes. Data points are 2x2 (empty boxes, 10000 trials per point), 4x4 (filled boxes, 10000 trials per point), 8x8 (empty circles, 10000 trials per point), 16x16 (filled circles, 10000 trials per point), 32x32 (empty diamonds, 10000 trials per point), 64x64 (filled diamonds, 5000 trials per point), 128x128 (empty up triangles, 5000 trials per point), 256x256 (filled up triangles, 2000 trials per point), 512x512 (empty down triangles, 500 trials per point), 1024x1024 (filled down triangles, 100 trials per point), approximation to infinite lattice (crosses).*

increasing p. This property makes it possible to use the percolation model as an illustrative tool to help explain a number of phase transition phenomena such as those shown in Table 3.2.

In the study of phase transitions or critical phenomena, the use of critical exponents has been most helpful in describing the behaviour at the critical point. Therefore, if we are to use the percolation model to describe other phase transition phenomena we must first extract the critical exponents for its own behaviour. The main technique of finding the critical exponents is via computer simulation, although for certain lattices and dimensions, exact results are known. Here we will temporarily restrict ourselves to site percolation on two-dimensional square lattice problems. Note that critical exponents generally depend only upon the dimension of the system, or lattice, under study and not on the particular lattice being used [Stanley1971].

Lattice	Site	Bond	Dimension
Honeycomb	0.6962	0.65271	2
Square	0.59275	0.50000	2
Triangular	0.50000	0.34729	2
Diamond	0.428	0.388	3
Simple Cubic	0.3117	0.2492	3
Body Centered Cubic	0.245	0.1785	3
Face Centered Cubic	0.198	0.119	3

Table 3.1: *Percolation thresholds for various two and three-dimensional lattices. Taken from [Stauffer1985]*

Phenomena or System	Transition
Flow of liquid in porous medium	Local/extended wetting
Spread of disease in a population	Containment/epidemic
Communication or resistor networks	Disconnected/connected
Conductor-insulator composite materials	Normal/superconducting
Discontinuous metal films	Insulator//metal
Stochastic star formation in spiral galaxies	Nonpropagation/propagation
Quarks in nuclear matter	Confinement/nonconfinement
Thin helium films on surfaces	Normal/superfluid
Metal-atom dispersions in insulators	Insulator/metal
Dilute magnets	Para/ferromagnet
Polymer gelation, vulcanization	Liquid/gel
The glass transition	liquid/glass
Mobility edge in amorphous semiconductors	Localized/extended states
Variable-range hopping in amorphous semiconductors	Resistor-network analog

Table 3.2: *Some applications of the percolation model. After [Zallen1983]*

3.2.1. Finding the critical exponents

Before one can find the critical exponents of any phase transition phenomena, one must first establish the critical point; for the percolation model this is the value of the percolation threshold, p_c. In addition one must also settle on a means of analysis. Here we will use computer simulation since the objective of this chapter is to describe computer architectures which will speed up such simulations. All results shown in this section are from actual simulations performed on a SUN3-160 engineering workstation using an additive feedback pseudorandom number generator like that described in Chapter 2. Later, results will be shown which are derived from the proposed architectures. The percolation model is easy to simulate since one merely assigns each site in the lattice as occupied, or ununoccupied, with probability, p, and then performs the

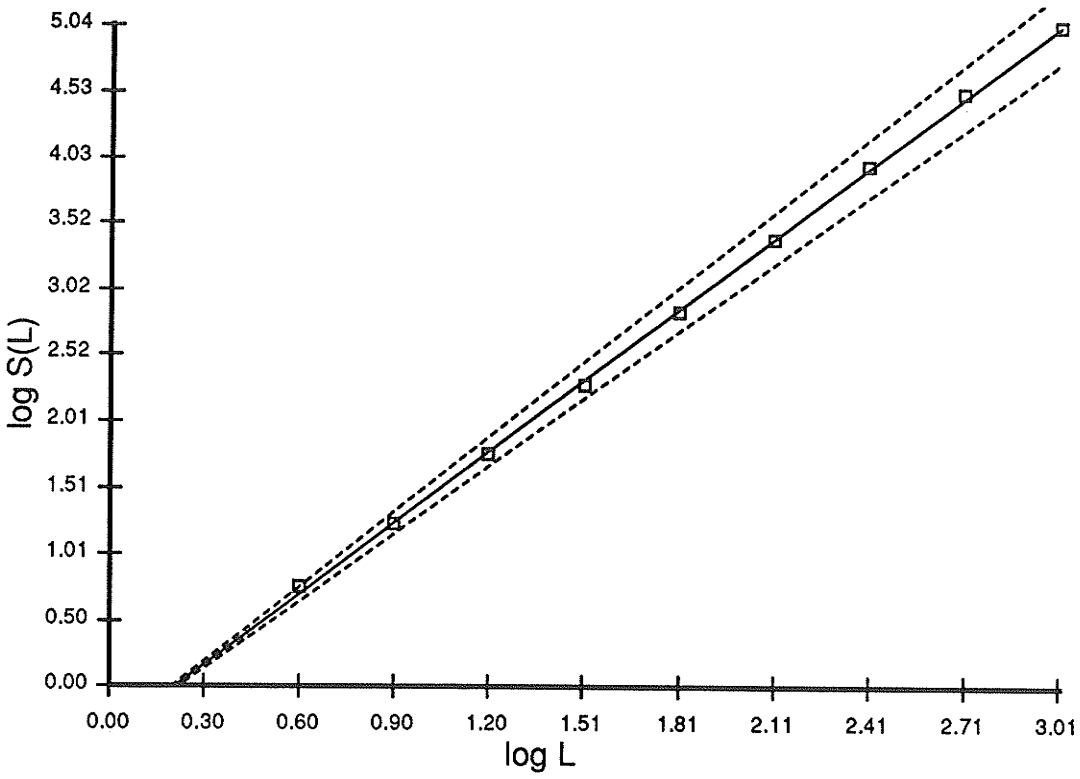


Figure 3.3 : *Size of $S(L)$ at p_c versus L.*

desired measurement (e.g. size of largest cluster, is there a spanning cluster?, etc...). Many of the two-dimensional problems discussed here have exact analytic solutions [Essam1978] but are nevertheless employed in percolation simulations since they are the easiest to understand. One should also note that many percolation problems, when considered in a higher dimensionality than two can only be analysed via computer simulation. We now proceed to show how three of the six common critical exponents for the percolation model can be found. The solution techniques are taken from [Kirkpatrick1978] and [Sur1976].

The percolation threshold, using the computer simulations described above, was found to be 0.5916 ± 0.0022. This is very close to other published values of 0.5928 [Stauffer1985].

The first critical exponent we will find describes the rate of growth of the largest cluster formed at $p = p_c$ as a function of lattice size. This is known as scaling theory. Here we define $S(L)$ to be the size of the largest cluster at the percolation threshold as a function of lattice size, L.[3.1] Note that since we can simulate only finite lattices the percolation threshold, p_c', is different from our reference point p_c for the infinite lattice. [Margolina1983] has shown that $S(L) \sim L^{1/\alpha}$. Therefore, if we plot log $S(L)$

[3.1] Here L denotes the length in one dimension of the lattice, i.e. the number of sites in a two-dimensional lattice of size L, is L^2.

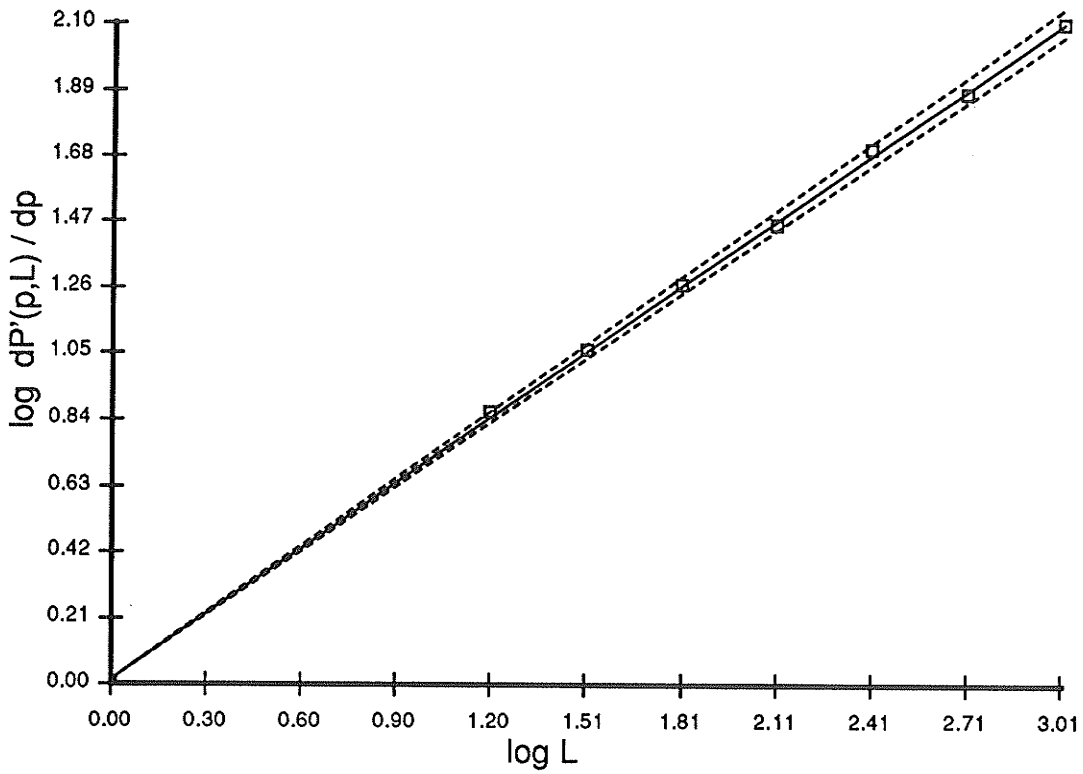


Figure 3.4 : *Log-log plot of* $\frac{dP'(p,L)}{dp}$ *versus* L.

versus log L as in Fig. 3.3 then $1/\alpha$ is the slope. It is known from an exact solution of scaling theory that $1/\alpha = 91/48 \approx 1.89$ [Stauffer1985]. Simulations on the SUN3-160 yielded a value of $1/\alpha = 1.798 \pm 0.093$.

The second critical exponent is more difficult to find. Recall that $P'(p,L)$ is the probability that percolation has occurred in a lattice of size L with a site probability of p. A plot of $P'(p,L)$ versus p was shown earlier in Fig. 3.2. We then use the scaling relation due to [Reynolds1978] which shows that

$$\frac{dP'(p,L)}{dp} = L^{1/\nu} \quad . \tag{3.1}$$

Finally, we plot $\frac{dP'(p,L)}{dp}$ versus L on a log-log plot as in Fig. 3.4 and extract the value of ν from the inverse of the slope. The simulations performed here determined that $\nu = 1.439 \pm 0.015$ as compared to the generally accepted value of 1.355 [Klein1978].

The last critical exponent which we will find concerns the percentage of sites in the largest cluster as a function of p. Define

$$R(p,L) = \frac{\textit{number of sites in largest cluster}}{\textit{number of sites in the sample}} \quad . \tag{3.2}$$

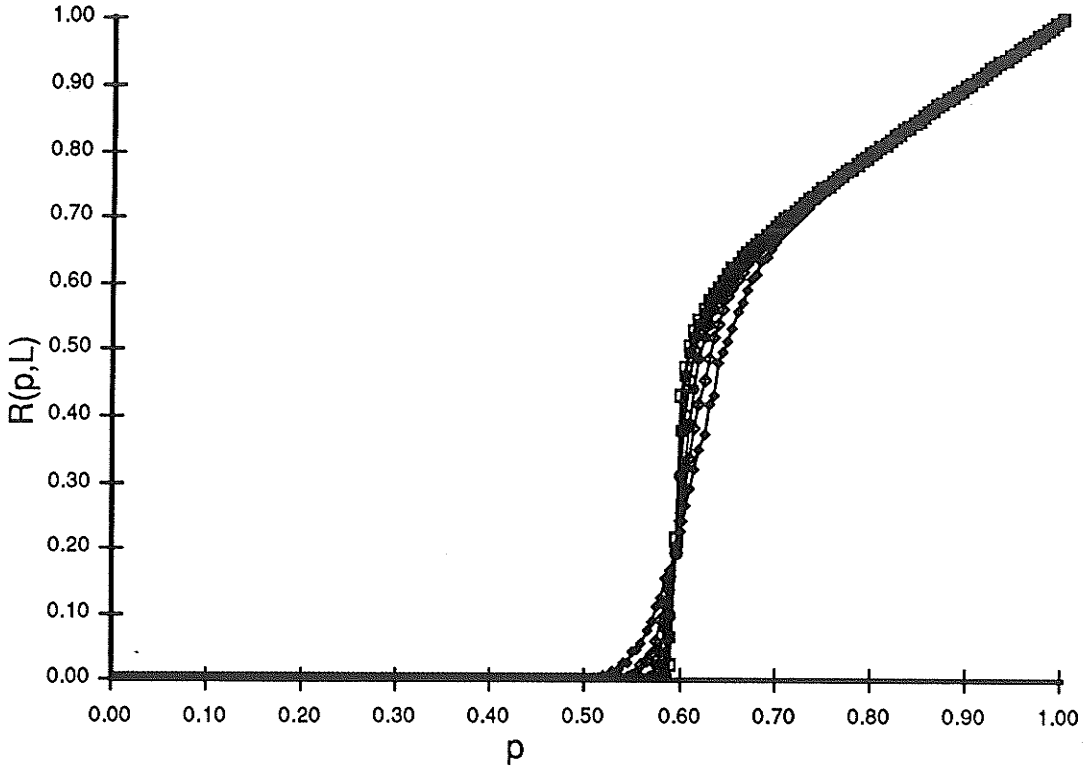


Figure 3.5 : *$R(p,L)$ versus p for various lattice sizes, data points are as in Fig. 3.2.*

In Fig. 3.5 we show $R(p,L)$ versus p for various lattice sizes. Notice that as the lattice size, L, increases the transition near p_c becomes more abrupt. The transition is rounded because of finite sampling effects. Here we use the rounding due to finite lattice size to extract the final critical exponent. A scaling relation due to [Fisher1971] states that

$$R(p,L) \sim L^{-\beta/\nu} X_1 \left[L^{1/\nu} \frac{p-p_c}{p_c} \right] . \tag{3.3}$$

where, X_1 is an appropriate scaling function of $L^{1/\nu}(p-p_c)/p_c$. Therefore, if we plot $R(p,L)\ L^{\beta/\nu}$ versus $X_1(L^{1/\nu}(p-p_c)/p_c)$ for various values of L, the value of β will correspond to the best fit of the curves near the percolation threshold. In Fig. 3.6 a least squares fit [3.2] to the curves of Fig. 3.5 yields a value of $\beta = 0.144$ as compared to the accepted value of 0.14 [Zallen1983].

Other common quantities in the percolation model for which critical exponents are often calculated are site correlation or spanning length (the maximum separation of

[3.2] Note that the curves are fit on a finite range of p surrounding the percolation threshold. Regions outside this area are not included in the curve fitting process.

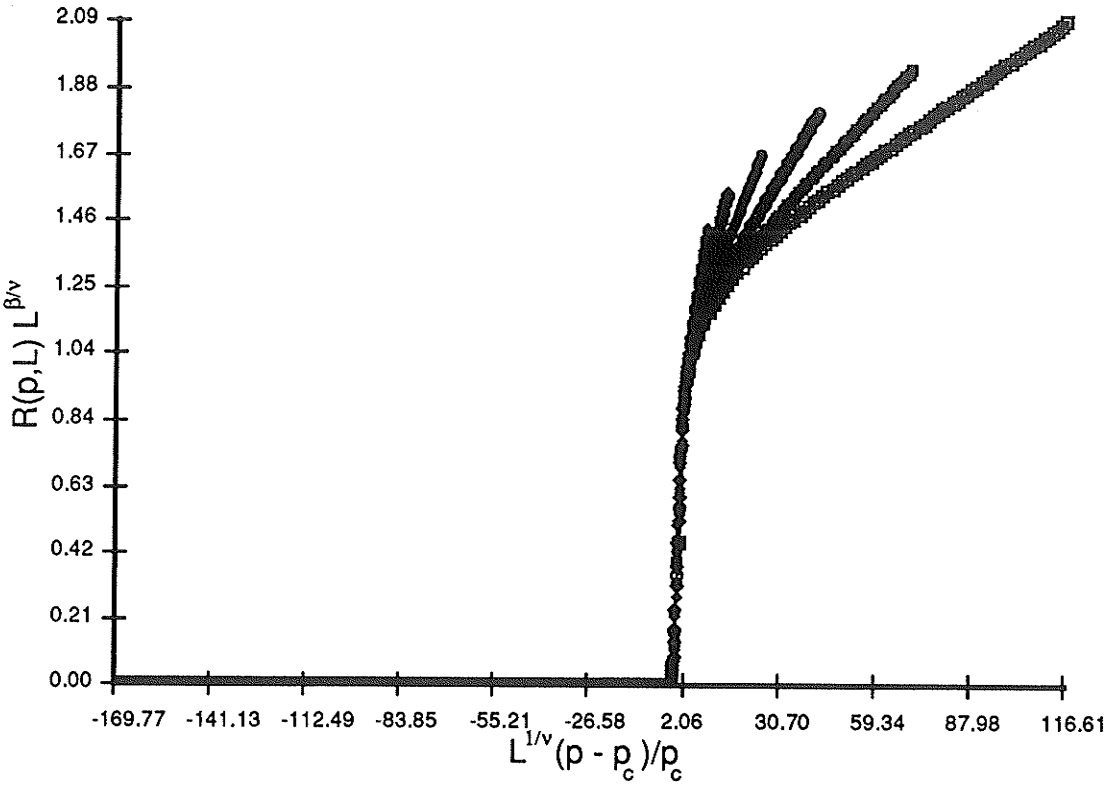


Figure 3.6 : *Best fit near p_c of $R(p,L)\ L^{\beta/\nu}$ versus $X_1(L^{1/\nu}(p-p_c)/p_c)$ for various lattice sizes.*

two sites in a cluster), pair connectedness (the probability that two sites separated by a given distance are members of the same cluster), and the conductivity (the conductance across a corresponding random resistor network). Here we have not considered these other quantities, but the three quantities which we do study are representative of the calculations which must be carried out in order to study the percolation model. It is not expected that any uncalculated critical exponent will deviate further from its known, or expected, value than those critical exponents which are calculated in this work.

3.2.2. Proposed Percolation Architecture

The computational work in any percolation simulation on a typical serial computer consists of the actual generation of the percolation lattice with site probability p and the calculation of the appropriate quantity of interest. To simulate percolation on a lattice we must generate a pseudorandom number, for each site with a given p, and occupy the site accordingly. This operation is repeated over the entire $N = L^2$ sites of the lattice. Therefore, we require at least $O(N)$ time to generate a single copy of the lattice. For the three critical exponents above we must calculate the probability of percolation and the size of the largest cluster at any probability p. To calculate whether the lattice has a percolating cluster and the size of the largest cluster requires no more than $O(N^2)$ time using the Hoshen and Kopelman cluster labelling algorithm

[Hoshen1976].

An obvious question to ask is which aspects of the percolation model simulation can be accomplished in parallel. While the portions of the algorithm which can be parallelised are fairly obvious[3.3] the best method to implement such a parallel computer is not. We note that the larger the lattice which can be simulated the greater the interest in the simulation. The largest simulation which has presently been carried out used a 160,000 × 160,000 square lattice [Rapaport1985].

The remaining problem is the calculation of the critical exponents. It is possible to build a special purpose computer which can both simulate the system and calculate the critical exponents for the percolation model. However, this is not necessarily the most expedient solution. The disadvantages of such an approach stem from the fact that the actual calculation of the critical exponents requires data memory and floating point calculations. However, it is well understood by anyone who has attempted to simulate the percolation model that very little time is actually spent calculating the desired exponents. Most of the computer time is used in generating new lattice configurations and grouping the occupied sites into clusters. Furthermore, operations using the clusters are generally very rapid given that site clustering has already occurred. Therefore, little is to be gained by building a computer dedicated solely to the calculation of critical exponents. Much can however be gained by building a device which can generate new lattices and form clusters quickly. This device would act as a special purpose *coprocessor* to a general purpose host computer and because of the nature of its specialised task could be made to operate very efficiently. Therefore, we will consider an implementation where a host computer will determine the actual critical exponents and do operations on clusters generated by a special purpose *percolation coprocessor*.

In order to speed the percolation simulation the architecture of Fig 3.7 is proposed. Each processor consists simply of a pseudorandom number generator (PRNG), comparator, and storage element, or site latch. Here we use the CA rule 30 based PRNG discussed in Chapter 2 with the CA-based PRNG connected so that it forms a long one-dimensional chain over the entire system. The site probability, p, is made available to each comparator by a system bus and the pseudorandom number from the PRNG is compared to it. Finally, the site latch is turned high ($> p$) or low ($\leq p$) accordingly. Each site in the lattice is assigned a unique processor. Therefore, after each clock cycle we have defined a new percolation lattice, as compared to at least $O(N)$ time for a serial updating technique. In addition, the time for a single clock cycle is quite small (≤ 50nsec.). Each simulation step on the serial computer is comparatively large (≥ 5μsec.) for a single site. In addition of course, to update the entire lattice the serial method must be applied N times; the present approach only once. The overall speed improvement is approximately $100N$. It should be noted that the size of the lattice which can be simulated is restricted by the number of processing

[3.3] Occupying sites based on the site probability.

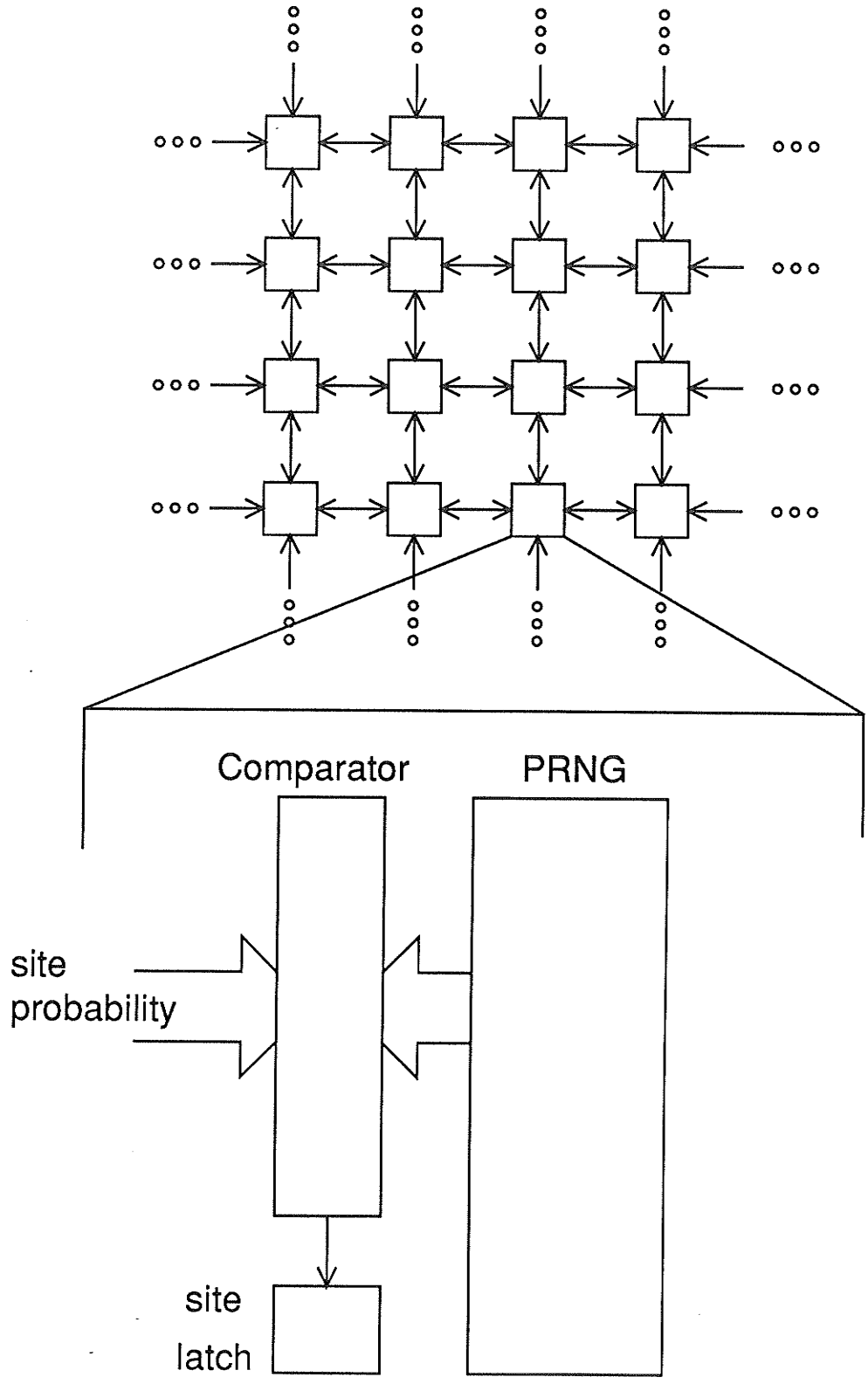


Figure 3.7 : *Basic percolation simulation architecture.*

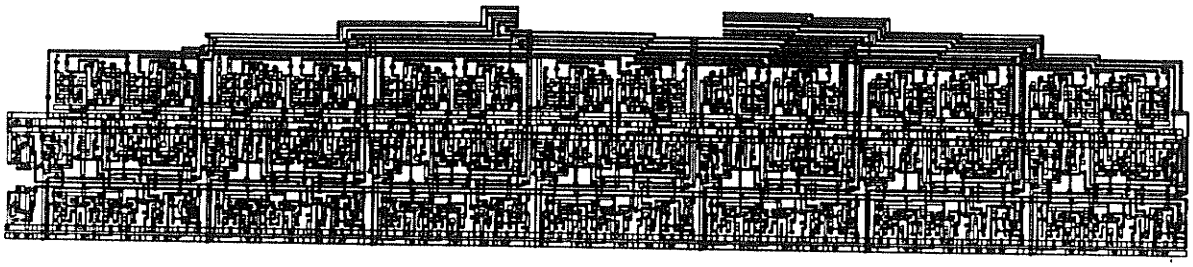


Figure 3.8 : *CMOS layout of 16 bit percolation site processor.*

sites available which is in turn dependent on the size of each processor.

The regularity of processing sites in such a percolation processor makes it an ideal candidate for VLSI implementation. The size of such a processor is directly dependent on register size. However, we can make estimates based on a fixed register size and scale up or down as appropriate for different register sizes. A 16 bit site processor is pictured in Fig. 3.8. The size of this processor is 0.838 mm^2 using the technology described in Chapter 2. Therefore it is possible to have 25 such processors simulating only a 5×5 lattice on a single 4.8×4.8 mm. die.[3.4] It is possible to implement the site processors in such a way as to be able to combine chips to form larger lattices. However, it is probably not realistic to consider employing a unique processor for each site in the lattice, if lattices larger than 1000×1000 are to be simulated. Therefore, we will restrict ourselves for the time being to lattices of $L \leq 1000$, i.e. those which have a unique processor for each site in the lattice. Later we will return to the problem of lattices larger than 1000×1000.

It is possible to use the proposed percolation architecture solely to dramatically increase the speed of updating the lattice. However, if we could calculate the size of the clusters and whether or not the largest cluster spans the lattice we would speed the simulation even more dramatically. It is possible to quickly group the occupied sites of the lattice into clusters if we superimpose the multiprocessor architecture of Fig. 3.9 onto the architecture of Fig. 3.7. Here we assign each processor a unique cluster number corresponding to its location in the lattice. For example, in Fig. 3.9 we have assigned processors in the first row to have values 0 to $L - 1$, the second row processors are assigned numbers L to $2L - 1$, and so on. This *percolation computer* operates as follows. First we utilise the underlying architecture of Fig. 3.7 to decide which sites are occupied. Occupied sites take their assigned cluster value while

[3.4] This technology (3 μm CMOS) available to us is not state of the art. Implementation using much more advanced technology would dramatically increase the number of site processors per chip. For example, on 1μm technology using a 10×10 mm die one could easily place over 1000 site processors.

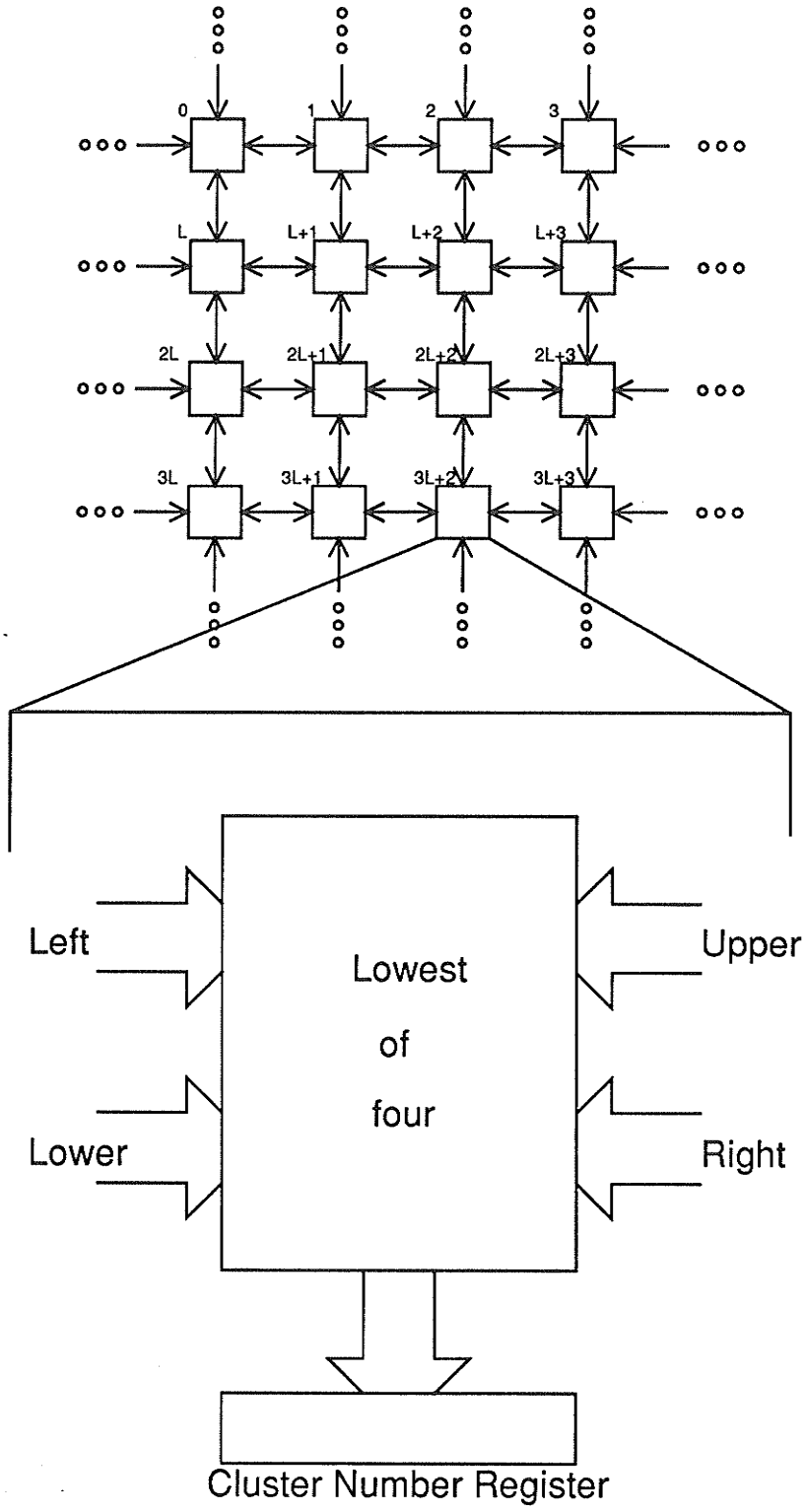


Figure 3.9 : *Architecture to group occupied sites into clusters.*

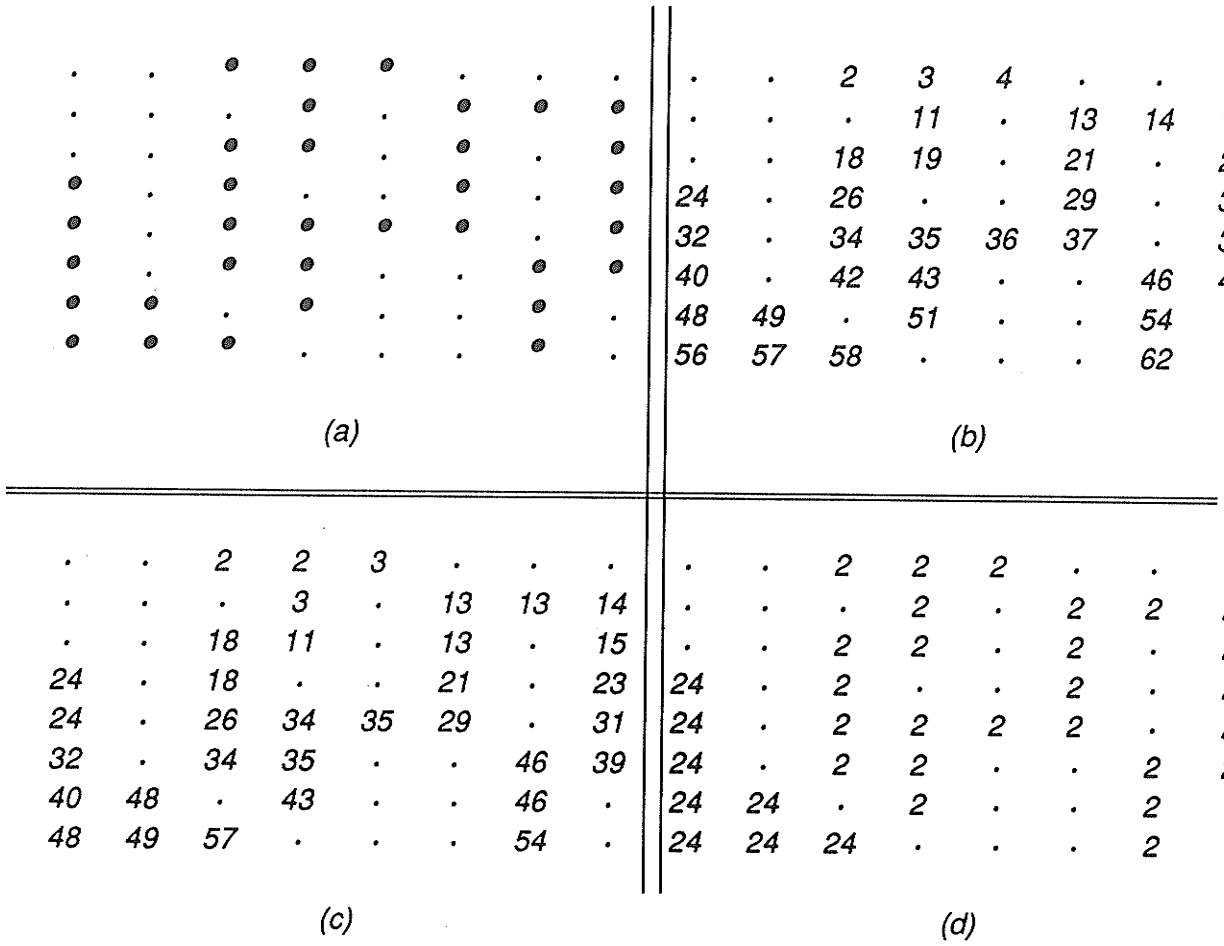


Figure 3.10 : *Operation of percolation computer on an 8 × 8 square lattice; (a) 8 × 8 square lattice with $p = 0.5313$. (b) initialised lattice with cluster number assignment. (c) cluster numbers after one update. (d) final cluster numbering assignment.*

unoccupied sites take on the value of ∞, or some other appropriately large number. We then proceed to synchronously update all sites according to the following algorithm. If a site is occupied, the next cluster value is selected as the lowest of its four neighbouring cluster values (remember we are presently considering only square two-dimensional lattices) and itself. For example, in Fig 3.10(a) we see an 8 × 8 lattice with $p = 0.5313$, Fig. 3.10(b) shows the same lattice initialised using the above cluster numbering assignment and in Fig. 3.10(c) we see the cluster numbers one synchronous update later. The synchronous updating procedure continues to take place until all sites belonging to the same cluster have had their cluster numbers merged together. The worst case time for this procedure would be $L\ (L - 1)/2$. The final cluster numbering configuration for Fig. 3.10(a) is shown in Fig. 3.10(d).

Determining whether or not the largest cluster is infinite is quite easy if we realise that a spanning cluster must be present both at the top and bottom of the lattice. Therefore, if any sites on the bottom of the lattice have a final cluster number less than

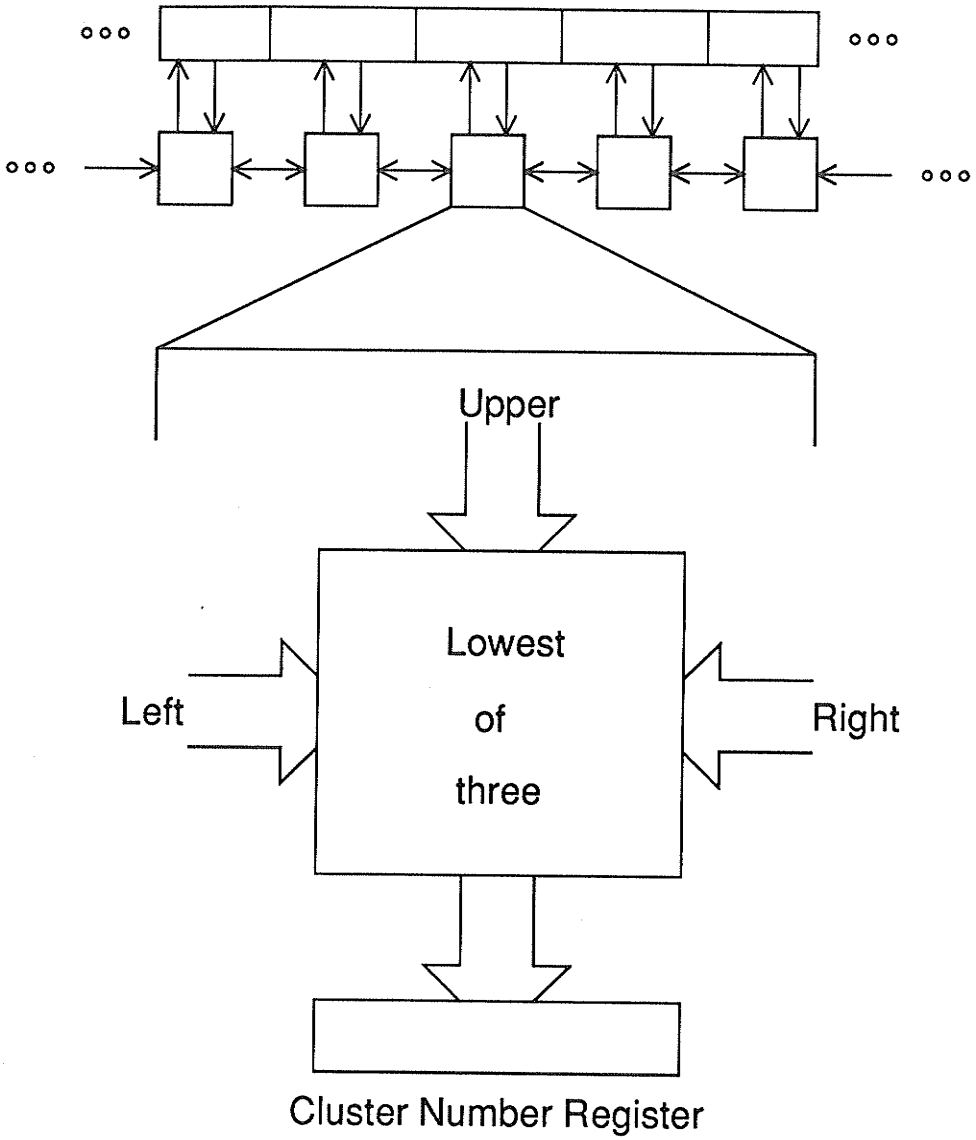


Figure 3.11 : *Architecture to group sites into clusters for row at a time lattice generation.*

L then we have a spanning cluster. Therefore, for a 1000×1000 lattice we can group the occupied sites into clusters and determine if a spanning cluster exists in at most 500,000 update steps. Measurements of the circuit shown in Fig. 3.8 have shown the operating speed to be at least 20 MHz. Thus, in about 25 milliseconds we can generate a 1000×1000 lattice, group the occupied sites into clusters, and determine if a spanning cluster is present.

To determine the size of the largest cluster is a much more difficult problem. However, it is possible for the host computer to offload the final cluster numbers from

the percolation computer and count the number of sites in each cluster. This remains a significant enhancement over serial computer simulation techniques since much of the time is spent grouping occupied sites into clusters. Other calculations for quantities such as pair connectedness and site correlation are also significantly faster since the clusters have already been formed. Finally, we note that is also possible to place processing elements which can perform cluster sizing calculations into the architecture. However, such processing elements are considerably more complex, especially since they require data memory, so they are not considered in this work.

3.2.3. Simulating Larger Lattices

As mentioned above we must presently restrict the lattice size to approximately 1000 × 1000 when considering an architecture where there is a unique processor corresponding to each lattice site. For larger lattices we can assign the processors of the percolation computer to correspond to unique sites on each row, or rows, of the lattice. Therefore, we now consider percolation problems on lattices up to 1,000,000 × 1,000,000 sites. To determine whether each site on a row of the lattice is occupied we can use the same technique as discussed above with respect to Fig. 3.7. Grouping the sites into clusters is not possible without keeping an entire history of the lattice. However, it is possible to dramatically assist the host computer. Here we use the scheme of Fig. 3.11 which keeps a copy of the previous lattice row. As before, we assign a unique cluster number to each site of the lattice. Note that we must assign different numbers to each row. Each site processor now determines a new cluster number based on the value of its upper, right, and left neighbours. This process continues for a maximum of L updates until all sites have been grouped into their respective clusters. The cluster numbers are then offloaded into the host computer and another row is determined and grouped into clusters. An example of this process is shown in Fig. 3.12 which implements this technique on the lattice of Fig. 3.10. In Fig. 3.12(a) we see the second row after it has been initialised. Figure 3.12(b) shows the second row cluster numbers after all sites have been grouped into clusters. Note that cluster numbers 2 and 13 refer to the same cluster but the percolation computer cannot know this yet since the merging of these two clusters occurs from the bottom up. This process continues until in Fig. 3.12(c) we see updating of row 5 where it is discovered that clusters 2 and 13 are the same cluster. As this row is offloaded the host processor must note that clusters 2 and 13 are the same cluster. Finally, we see in Fig. 3.12(d) the full lattice as it would be received and stored by the host processor.

3.2.4. Simulation Results for the Percolation Computer

Simulations of the percolation computer were carried out and yielded the following results. The percolation threshold was found to be 0.5915 ± 0.0023. The rate of increase of $S(L)$ is shown in Fig. 3.13, from which we can see that $1/\alpha = 1.809 \pm 0.096$. The probability of percolation versus site probability using the percolation computer is shown in Fig. 3.14. Figure 3.15 shows a log-log plot of $\frac{dP'(p,L)}{dp}$ versus L, from which we can calculate that $\nu = 1.434 \pm 0.030$. Figure 3.16

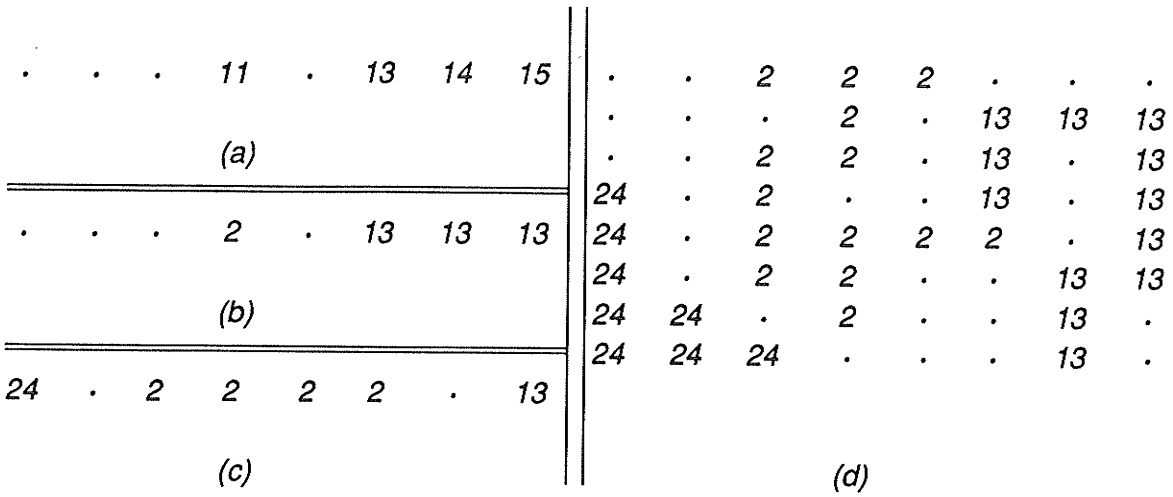


Figure 3.12 : *Operation of row at a time percolation computer on lattice of Fig. 3.10; (a) initialised second row. (b) second row after cluster-numbering completed. (c) fifth row after cluster-numbering completed. (d) fully cluster-numbered lattice.*

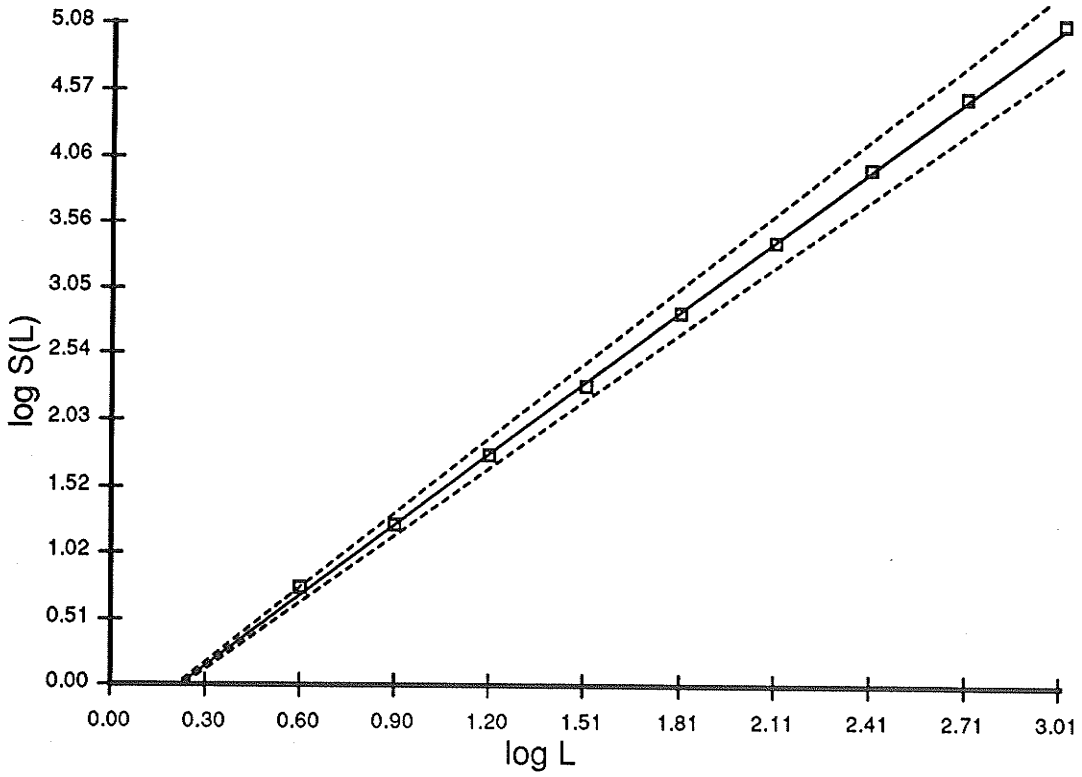


Figure 3.13 : *Size of $S(L)$ at p_c versus L using the percolation computer.*

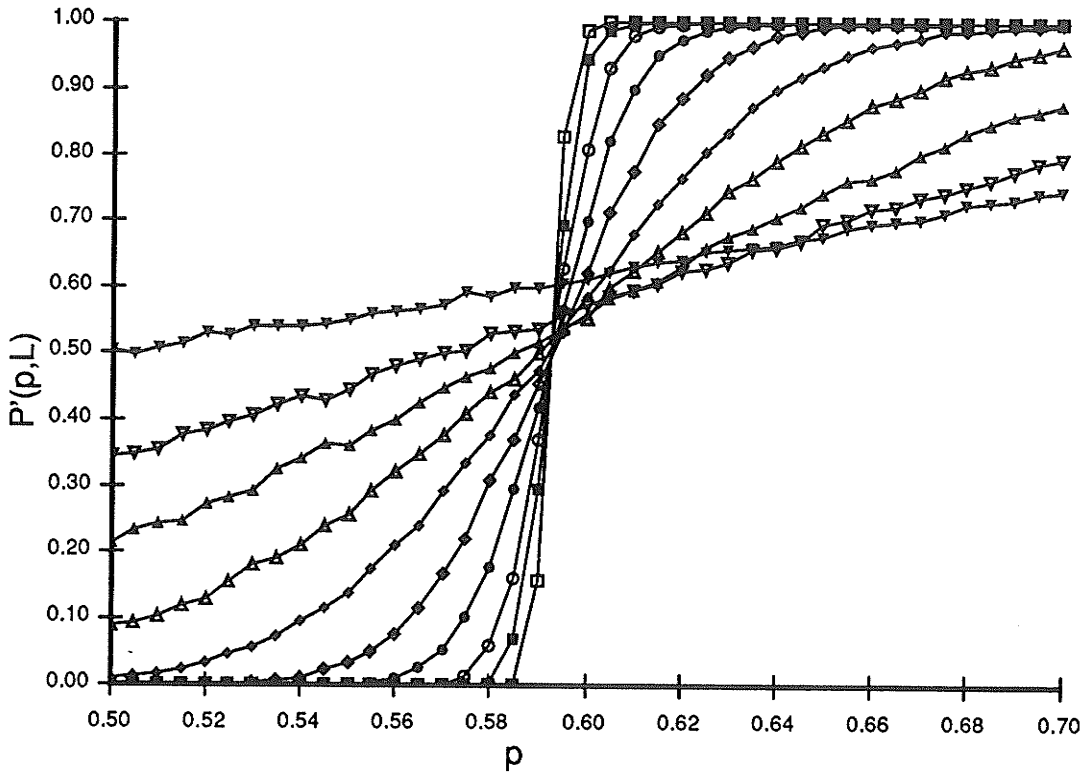


Figure 3.14 : *Probability of percolation versus site probability for various lattice sizes using the percolation computer. Data points are 2x2 (filled down triangles, 10000 trials per point), 4x4 (empty down triangles, 10000 trials per point), 8x8 (filled up triangles, 10000 trials per point), 16x16 (empty up triangles, 10000 trials per point), 32x32 (filled diamonds, 10000 trials per point), 64x64 (empty diamonds, 5000 trials per point), 128x128 (filled circles, 5000 trials per point), 256x256 (empty circles, 2000 trials per point), 512x512 (filled boxes, 500 trials per point), 1024x1024 (empty boxes, 100 trials per point).*

shows the the fraction of all occupied sites which are in the largest cluster versus site probability. Using the scaling relation of Eqn. 3.3, the best fit is for $\beta = 0.145$, as shown in Fig. 3.17. The results are summarised in Table 3.3. There is a small discrepancy between the results which have been calculated here and those which have been published elsewhere. However, there is close agreement between the results for the percolation computer and the standard percolation simulation using the additive feedback PRNG. Therefore, we can conclude that the parallel percolation computer yields the same critical exponents as a standard serial percolation simulation. The discrepancies between the critical exponents calculated here and those published elsewhere may be due to several factors such as smaller register size (here 16 bits was used) and a smaller number of samples. However, it is encouraging that percolation simulations using the proposed percolation computer and a standard serial

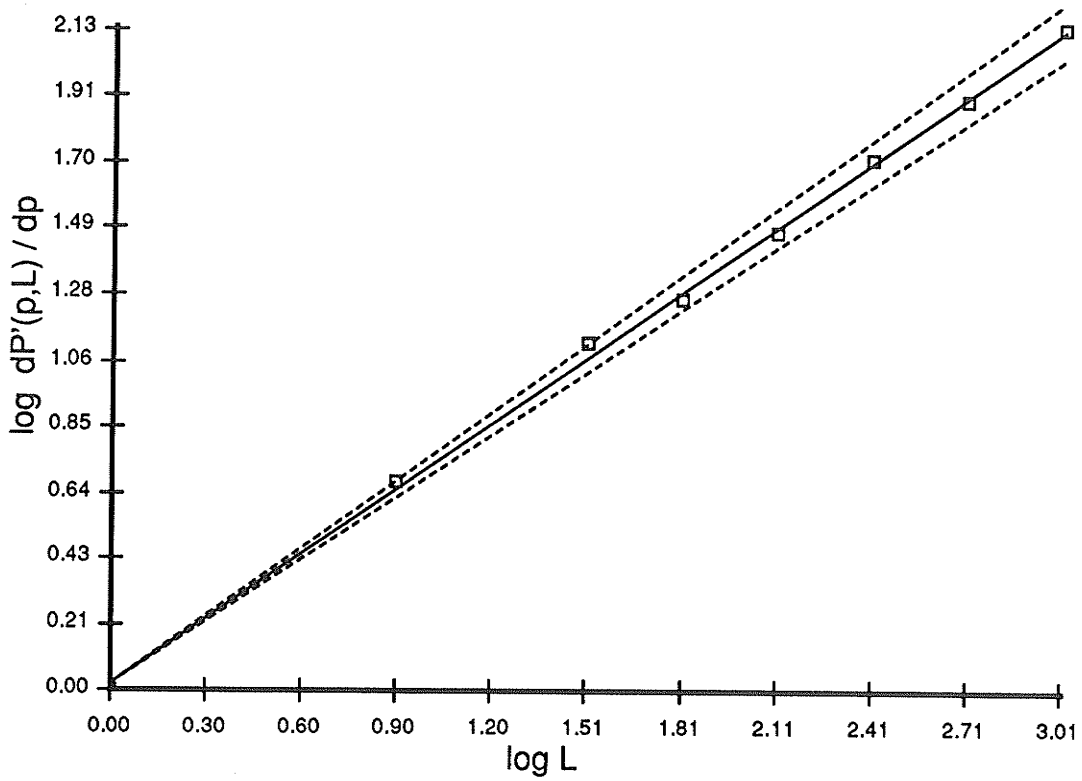


Figure 3.15 : *Log-log plot of* $\frac{dP'(p,L)}{dp}$ *versus L using the percolation computer.*

computer method yielded the same results.

In Chapter 2 it was indicated that site spacing could be used to improve the quality of a CA rule 30 based PRNG. Therefore, it would be natural to test the effects of site spacing on the critical exponent values produced by the percolation computer. Simulation shows that there is no appreciable change in critical exponent value if site spacing is used. Thus, it would appear that it is possible to construct the percolation computer without site spacing in the PRNG. No tests, other than very cursory ones, were made using the other CA-based PRNGs discussed in Chapter 2.

It would be much more expensive in terms of both area and time to use any PRNG other than the CA based ones. In addition, one can see that, because of the vast number of PRNGs required, the topological regularity of the CA approach provides a very clear advantage. Finally we note that it has been found that standard LFSR-based and some multiplicative congruential PRNGs are inadequate for Monte Carlo simulations [Parisi1985] since they do not produce correct critical exponents. We observe that the critical exponents calculated using the CA-based percolation computer provided reasonably correct values.

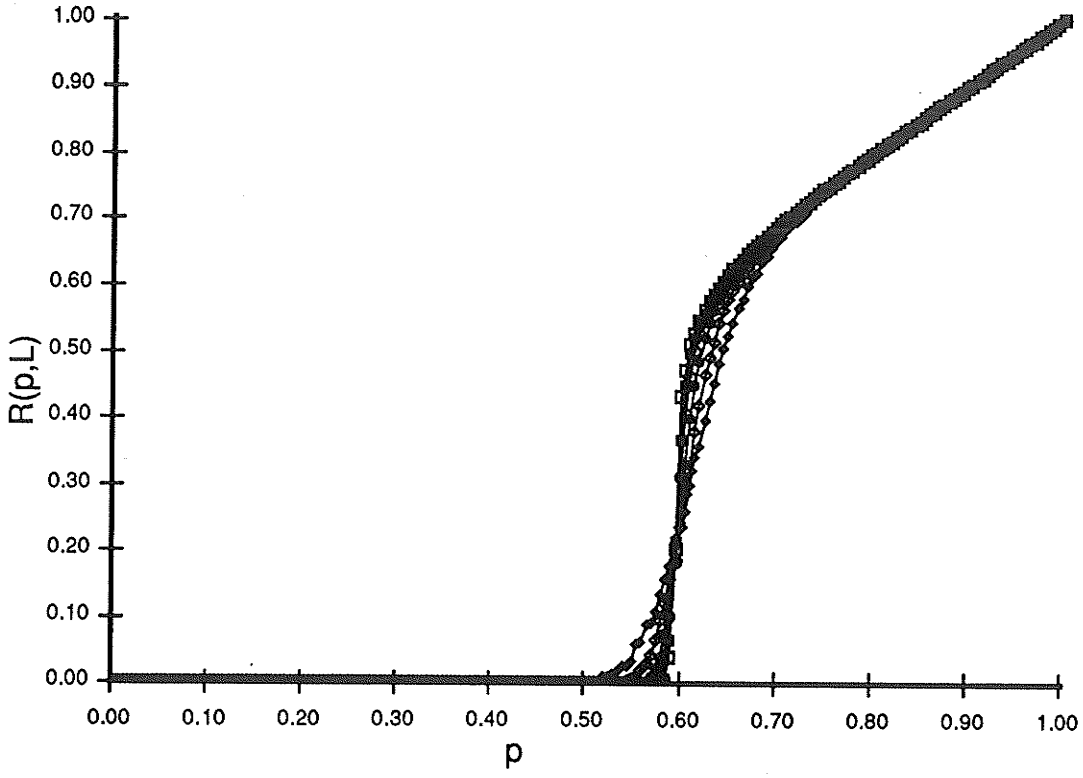


Figure 3.16 : *$R(p,L)$ versus p for various lattice sizes using percolation computer.*

3.2.5. Using Renormalisation on the Percolation Computer

Extensions to the percolation architecture could include the use of renormalisation group principles [Wilson1975] to extrapolate the infinite lattice critical exponents. The basic concepts required to implement renormalisation as applied to percolation are quite easy. Essentially we slowly integrate out small scale fluctuations and obtain information on successively larger and larger scales. This is done by replacing a small block of sites on the lattice with one site representing gross, or average, behaviour. For example, in majority rule renormalisation, a block of 3×3 sites is represented by one occupied site if the majority of sites are occupied and an unoccupied site if the majority of sites are not occupied. This procedure is repeated many times progressively reducing the lattice size by a factor of l for each renormalisation, where l is the size of the block of sites being replaced by a single site. The result is that for $p > 0.5$ the new p_1 representing, the density on the renormalised lattice, moves quickly towards a value of 1.0, while for $p < 0.5$ the new p_1 moves towards 0.0. However, for $p = 0.5$ the new p_1 will also equal 0.5. This critical value of $p = 0.5$ derives simply from the majority renormalisation rule and is not associated with the critical percolation value. The critical exponents are extracted from the rate at which the value of p_1 moves towards 1.0 or 0.0. The problem here is that for many lattices simple majority rule renormalisation is not adequate to extrapolate infinite lattice behaviour. In the above example we saw that for $p > 0.5$ the value of p_1 moved quickly towards 1.0.

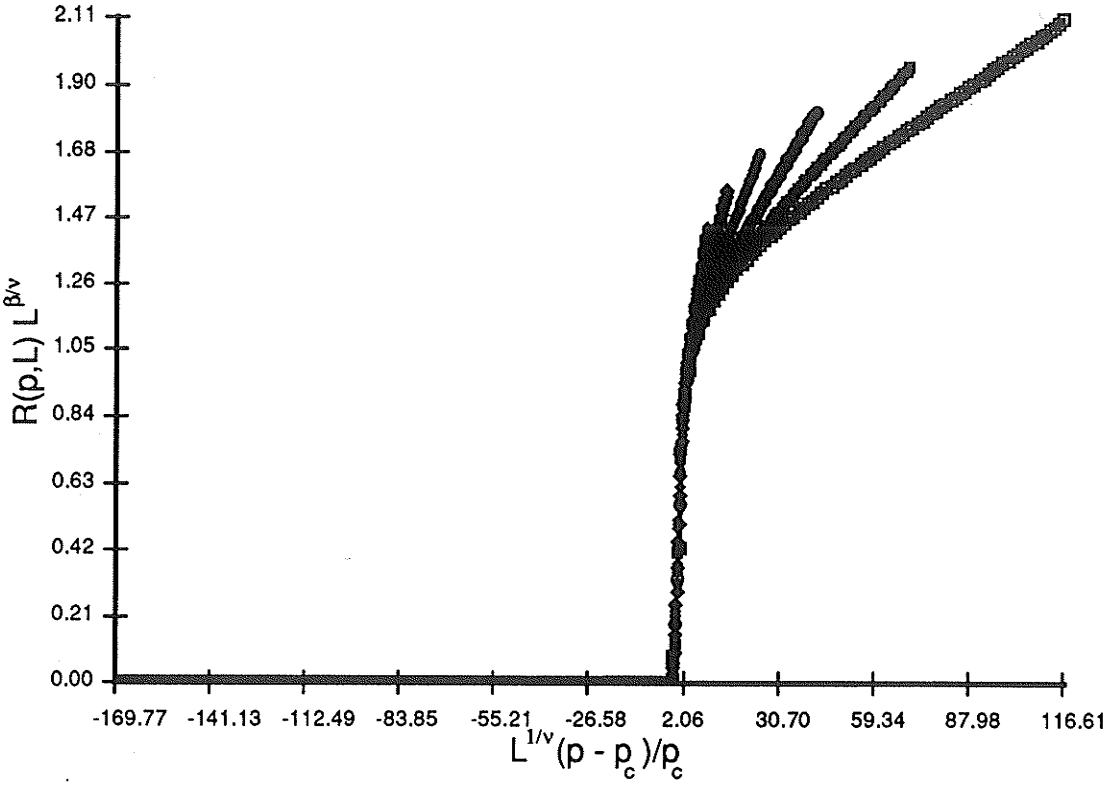


Figure 3.17 : *Best fit of curves from Fig. 3.16 using the scaling relation of Eqn. 3.3.*

Exponent	SUN	CA	Others	Reference
p_c	0.5916 ± 0.0022	0.5915 ±0.0023	0.5928	[Stauffer1985]
$1/\alpha$	1.798 ± 0.093	1.800 ± 0.096	1.89	[Stauffer1985]
ν	1.439 ± 0.015	1.434 ± 0.030	1.35	[Zallen1983]
β	0.144	0.145	0.14	[Zallen1983]

Table 3.3: *Percolation critical exponents, SUN refers to standard serial computer percolation simulations done for this work, CA refers to simulation results for percolation computer, others refers to representative results which has been reported elsewhere.*

Thus, for $p = p_c = 0.5928$ on the square lattice p_1 will move towards 1.0 and it is not possible to extract critical behaviour since p_1 is not equal to p_c. Therefore, another renormalisation rule is required if we are to study critical behaviour for site percolation on a square lattice using renormalisation techniques. For example, [Kirkpatrick1977] studied renormalisation on a square lattice by replacing a block of sites with an occupied site only if a spanning cluster, or connecting path, existed in the block. [Reynolds1977] and later in [Reynolds1978] utilised a position-space renormalisation procedure whereby a block of 2^d sites was replaced by a single site and d bonds,

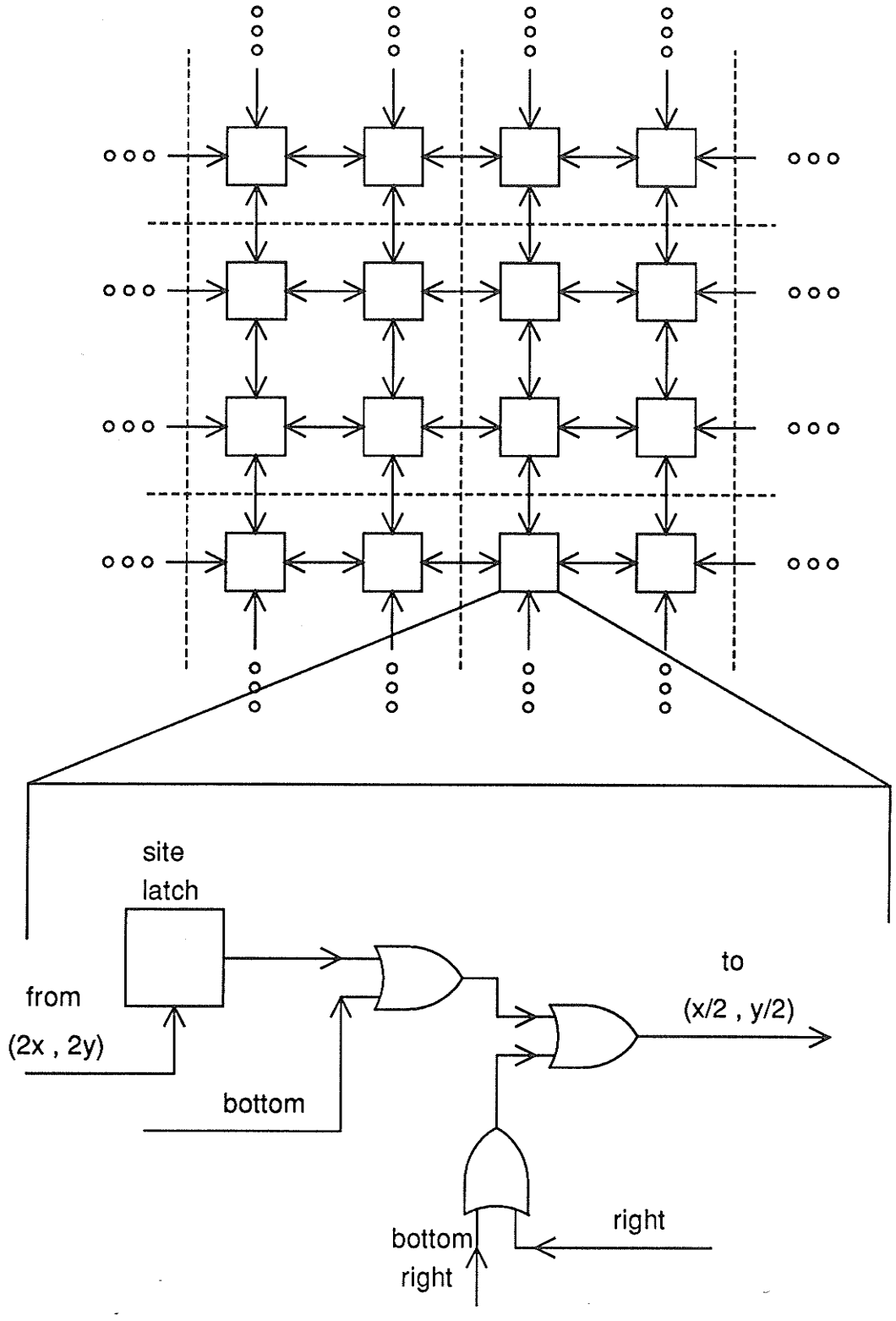


Figure 3.18 : *Renormalisation architecture operating on 2 × 2 blocks using a renormalisation rule due to [Kirkpatrick1977].*

requiring that the d bonds reflect the connectivity of the block which it is replacing.

In any case we see that construction of hardware to implement any renormalisation procedure, other than the simple majority rule case, requires significant processor resources. An example of a simple renormalisation group architecture is shown in Fig. 3.18. Here we implement the renormalisation rule of [Kirkpatrick1977]. For simplicity we use a block size of 2×2. To determine whether an infinite cluster exists in a 2×2 block merely requires checking if each row has an occupied site. A renormalised site representing sites (x,y), $(x+1,y)$, $(x+1,y+1)$, and $(x,y+1)$ in the old lattice will be stored in position $(x/2,y/2)$ in the new $L/2 \times L/2$ lattice, necessitating a shift to the left and up by $x/2$ and $y/2$ site processors. This will require additional shifting hardware at each site processor. Finally, we assign cluster numbers to each occupied site in the new lattice and invoke the site clustering process. As larger blocks or more complicated renormalisation rules are considered the associated computing hardware becomes considerably more complex. Thus, a percolation computer implementing renormalisation will not be further considered in this work. However, we note that if a percolation computer is to be constructed which itself calculates the critical exponents, it is probably best to use a renormalisation approach to quickly reduce the amount of cluster data which must be processed and offloaded to the host computer.

Another extension to the percolation computer is the inclusion of different lattices and dimensions other than the two-dimensional square lattice which we considered here. To include other lattice types, for example the triangular or honeycomb lattices, one need merely increase the connectivity of the site processors to account for the increased number of neighbours. Otherwise the method of operation is precisely the same. Similarly for higher dimensions one need merely increase the neighbour connections at each site processor to account for the increased neighbour set. No simulations were performed on percolation operating on different lattice types or higher dimensions since it is not expected that the correctness of the percolation computer will be affected by having more neighbours. We do not expect the computer time for simulations on the percolation computer to increase dramatically as the neighbour set increases.

3.3. THE ISING MODEL

We now turn our attention to the equilibrium statistical mechanics of d-dimensional Ising models. The Ising model is perhaps the most well known of statistical mechanical models which exhibit a phase transition. It was first introduced by Ising in 1925 [Ising1925].[4.5] The model was initially used to describe the behaviour of a ferromagnet near the Curie temperature.[4.6] However, it was quickly found that the Ising

[4.5] Sometimes the model is referred to as the Lenz-Ising model since the actual model was first introduced by Lenz [Lenz1920] in 1920. However, he did not calculate any properties of the model and general practice has become to refer to the model as just the Ising model.

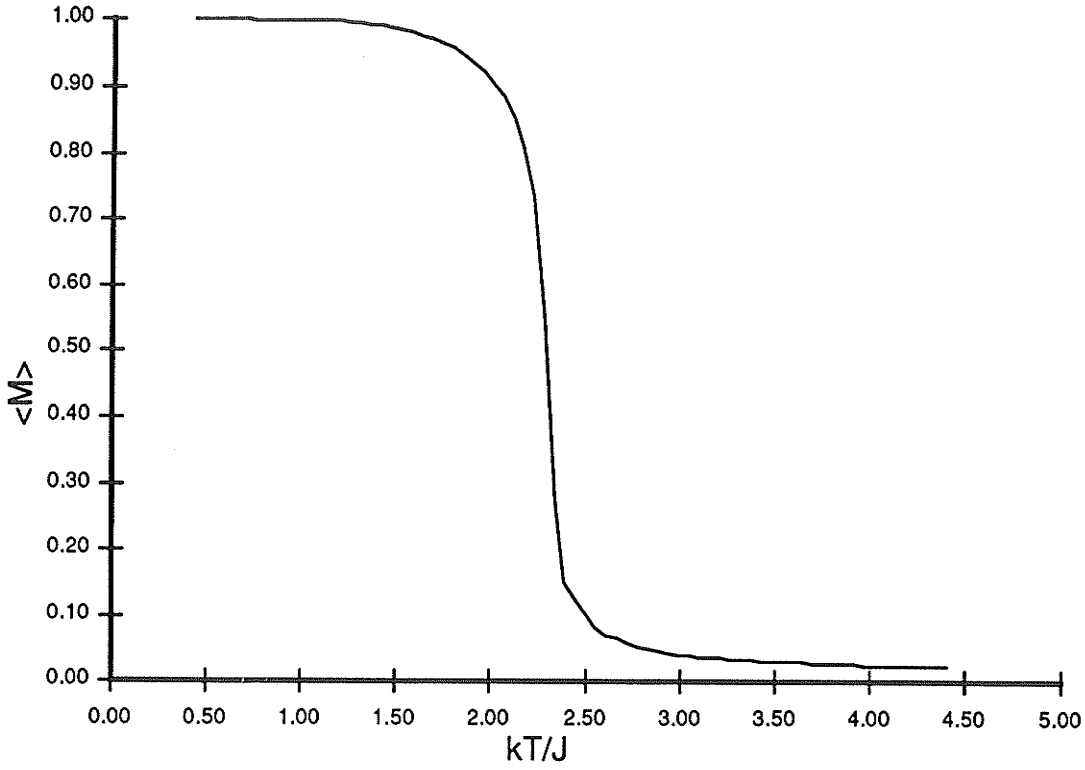
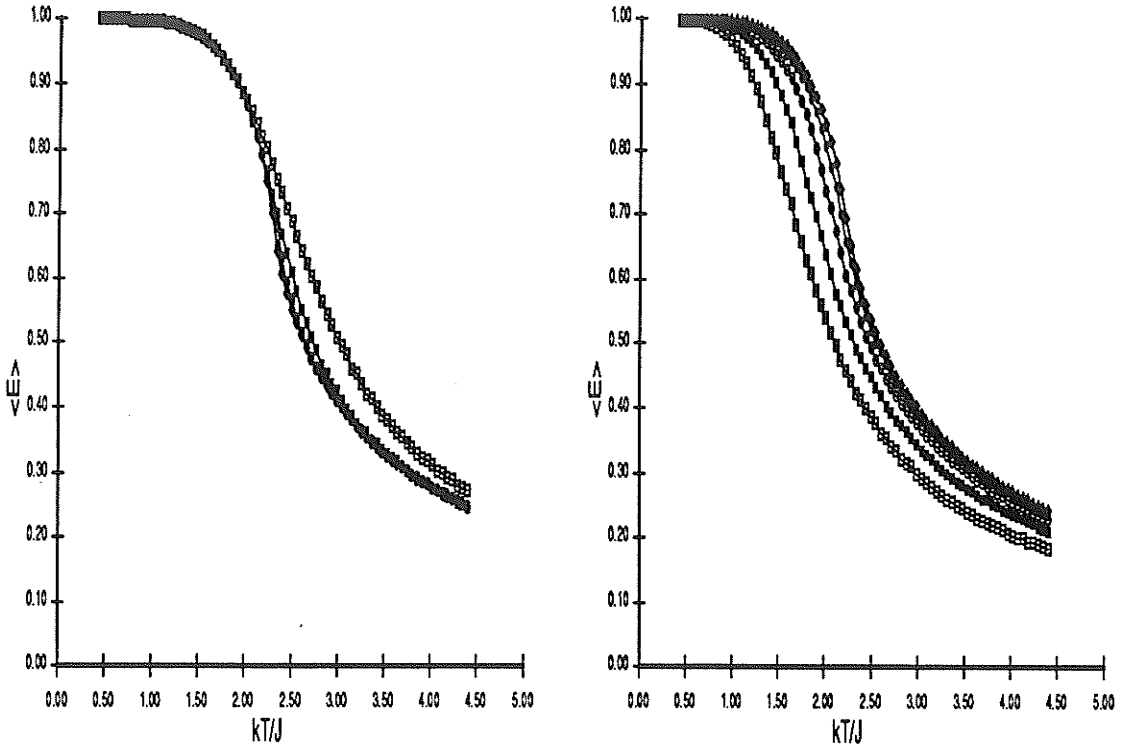


Figure 3.19 : *Magnetisation versus kT/J on a 64×64 two-dimensional square lattice.*

model could be used to describe and calculate many other physical properties. Today the Ising model is considered to be a paradigm for a wide variety of model systems in computational statistical mechanics [Binder1979], [Vichniac1984], [Kirkpatrick1985]; it is of central importance in the study of universality properties in critical phenomena, and in general of phase transitions in statistical-mechanical systems.

The essential concept of the Ising model is the description of the interaction of a set of atomic magnetic moments, or *spins*, arranged on a regular ferromagnetic lattice in d-dimensions. Here we define a positive, or up, spin to have the value of +1 and a negative, or down, spin to have value −1. The energy of two neighbouring spins is $-J$ if they are in the same direction and $+J$ if the spins are pointing in opposite directions, where J is a coupling constant. Therefore, the energy added to the total system energy by two adjacent lattice sites is $-J\ s_i s_j$, where s_i represents the spin at lattice site i. In a system which is subjected to a positive magnetic field, positive defined as pointing up, each spin will have an additional energy of $+H$ for down spins and $-H$ for

[4.6] At the Curie temperature a ferromagnet exhibits a phase transition from a paramagnet to a ferromagnet, much like the condensation of steam into water at the boiling point. For example, an iron ferromagnet is no longer magnetic at temperatures greater than 1043°K.



(a) (b)

Figure 3.20 : *Normalised energy versus kT/J for (a) periodic boundary conditions. (b) free boundary conditions. Data points are 4 × 4 (empty boxes), 8 × 8 (filled boxes), 16 × 16 (empty circles), 32 × 32 (filled circles), 64 × 64 (empty diamonds).*

up spins. Interaction between spins only occurs between nearest neighbours on the lattice which gives rise to a Hamiltonian, or total system energy of

$$E = -J \sum_{i,j} s_i s_j - H \sum_i s_i \tag{3.4}$$

where the sum over i,j includes neighbouring spins only.

The probability of finding adjacent lattice sites in a state $[s_i s_j]$ is given by a Boltzmann distribution,

$$P(s_i s_j) = Z^{-1} e^{-K s_i s_j} \tag{3.5}$$

where $K = \frac{-J}{kT}$, k is Boltzmann's constant, T is the absolute temperature, and the normalisation factor $Z = 2e^{K} + 2e^{-K}$. Therefore, at high temperatures the value of K is small and the alignment of spins is arbitrary while for lower temperatures K is much larger and the spins tend to align. When we consider a system with N spins the

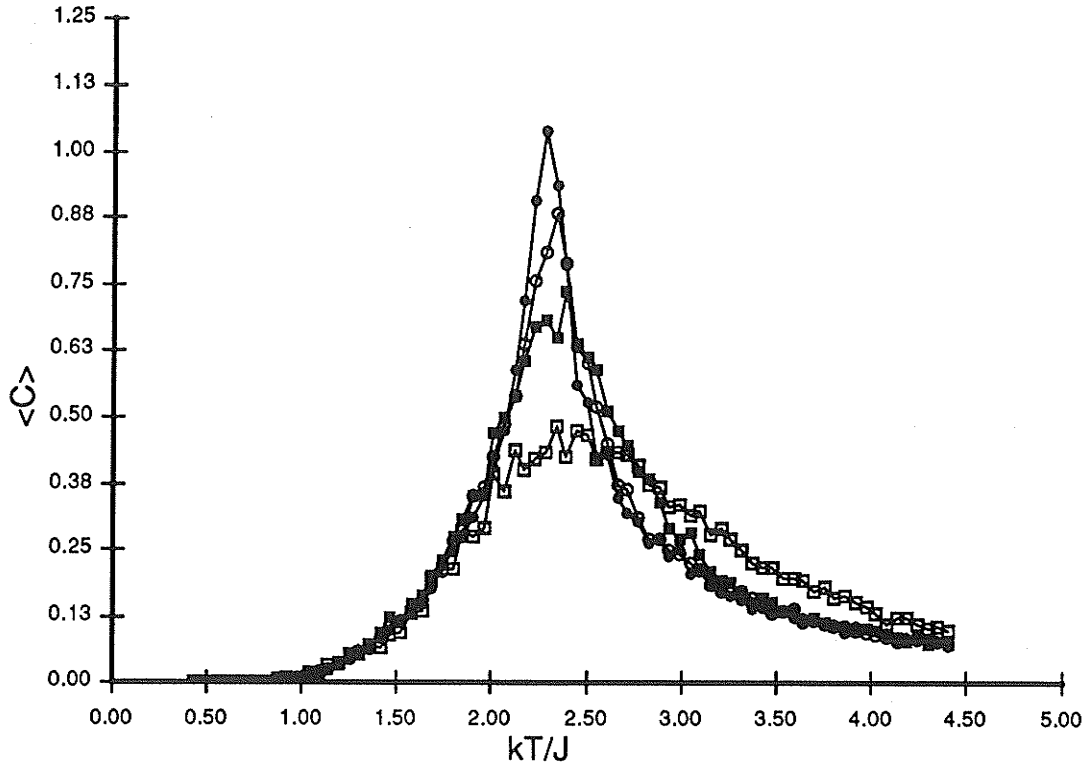


Figure 3.21 : *Specific heat versus kT/J for various size lattices with periodic boundary conditions.*

probability of state $S = [s_1, s_2, \cdots s_N]$ can be shown to be

$$P(S) = Z^{-1} e^{\frac{-E}{kT}} \tag{3.6}$$

where the normalisation factor Z is now the trace sum over all states S, i.e.

$$Z = Tr_{[S]}\, e^{\frac{-E}{kT}} \quad . \tag{3.7}$$

For J positive and $d \geq 2$, a phase transition occurs at a temperature $T=T_c$,the critical temperature (also called the Curie temperature), below which all the spins in the lattice tend to align with one another.

We define the spontaneous magnetisation of the lattice to be

$$M = \frac{1}{N} \sum_{i=1}^{i=N} s_i \quad . \tag{3.8}$$

The state of the lattice is constantly changing in time as thermal effects cause spins to change direction. For an arbitrary temperature T, the equilibrium system has an expected magnetisation, or net spin value, <M>, given by

$$<M> = \sum_j M_j \, P(S_j) \tag{3.9}$$

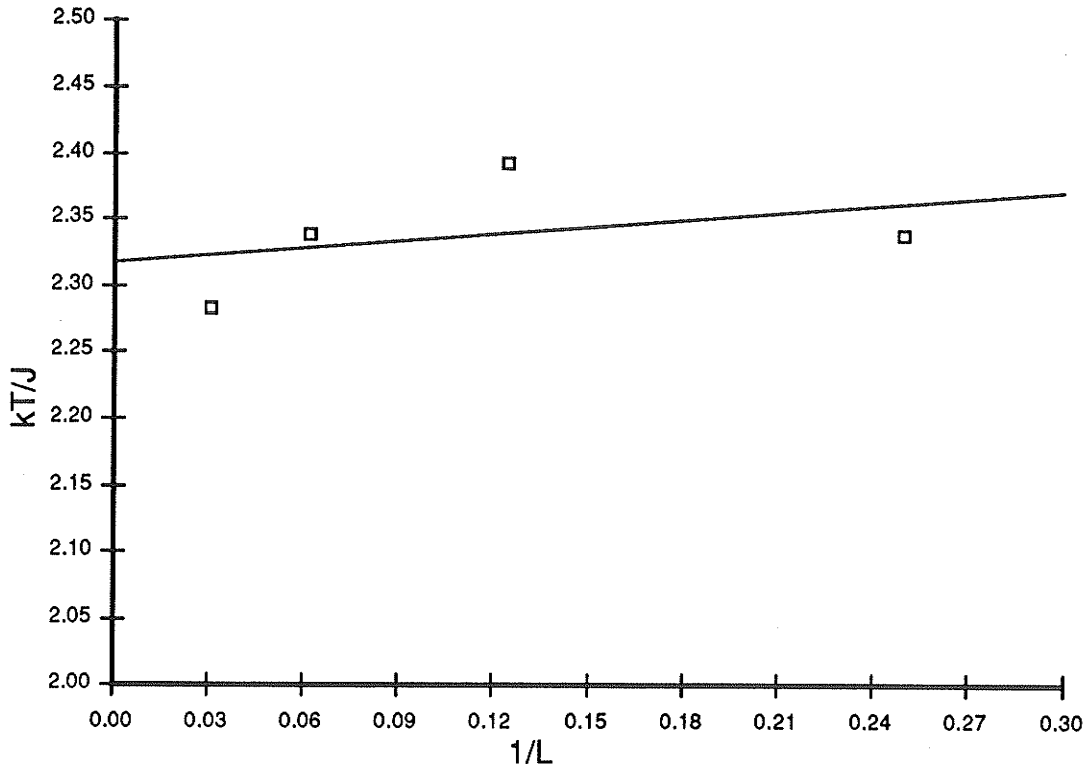


Figure 3.22 : *Extraction of $kT_c(\infty)/J$ using the specific heat. The data points give the temperature for the maximum specific heat.*

where the summation extends over all possible configurations of spins of the lattice, and M_j is the magnetisation of lattice state S_j. Therefore, to calculate the expected magnetisation, or any other observable, of a lattice we must determine the probability over all states of the lattice.

For a system with only two spins it is easy to calculate the total system energy since only four states must be considered. However, the problem explodes as the lattice size increases so that even for a simple two-dimensional 10 x 10 lattice there are 2^{100} possible configurations to consider. Therefore, one cannot calculate all the state probabilities for even small systems. One approach that immediately comes to mind is to chose states at random and then estimate the sum. However, we know from thermodynamic considerations that the distribution of states will be sharply peaked around the minimum energy configuration [Landau1968]. Therefore, there are many states which are highly improbable and contribute little to the dynamics of the system. Using random sampling will consider all states with equal probability and so we will consider many states which do not make a significant contribution to the system. Thus, for large N this method is not very efficient.

A more efficient technique involves an *importance sampling* [Metropolis1953] of configurations. This is performed by Monte Carlo methods [Binder1984], according to their total energy which determines their probability of occurrence, or weighting in

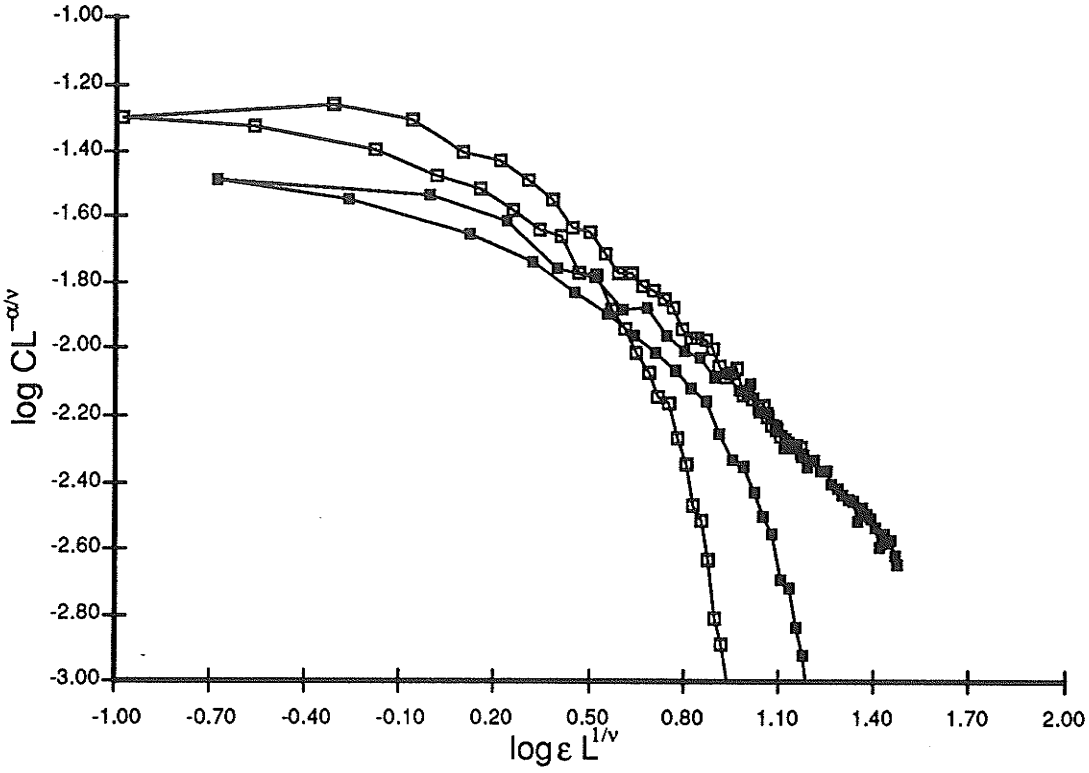


Figure 3.23 : *Finite-size scaling plot of the specific heat of lattices with periodic boundary conditions. Here* $\varepsilon = (T - T_c(\infty)) / T_c(\infty)$ *for* $T > T_c(\infty)$ *and* $\varepsilon = (T_c(\infty) - T) / T_c(\infty)$ *for* $T < T_c(\infty)$. *Data points for* $T > T_c(\infty)$ *(right hand curves) and* $T < T_c(\infty)$ *(left hand curves) line up with slope -1.0.*

Eqn. 3.9.

The mean value of an observable L at the temperature T is given by

$$<L> = \frac{\sum_{S'} L_\mu e^{\frac{-E_\mu}{kT}}}{\sum_{S'} e^{\frac{-E_\mu}{kT}}} \qquad (3.10)$$

where the sum need only be taken over the importance sample S' of configurations and the subscript $\mu \in S'$ refers to the specific sample.

A procedure to determine a representative set of configurations is given by the Metropolis algorithm [Metropolis1953]. In two dimensions the algorithm begins at the upper-left corner of the 2-D array and progressively updates the spins as it proceeds to the bottom-right corner. At any given stage in the process, the new spin value is decided stochastically; the local transition energy ΔE for spin $s_i \rightarrow -s_i$ is determined.

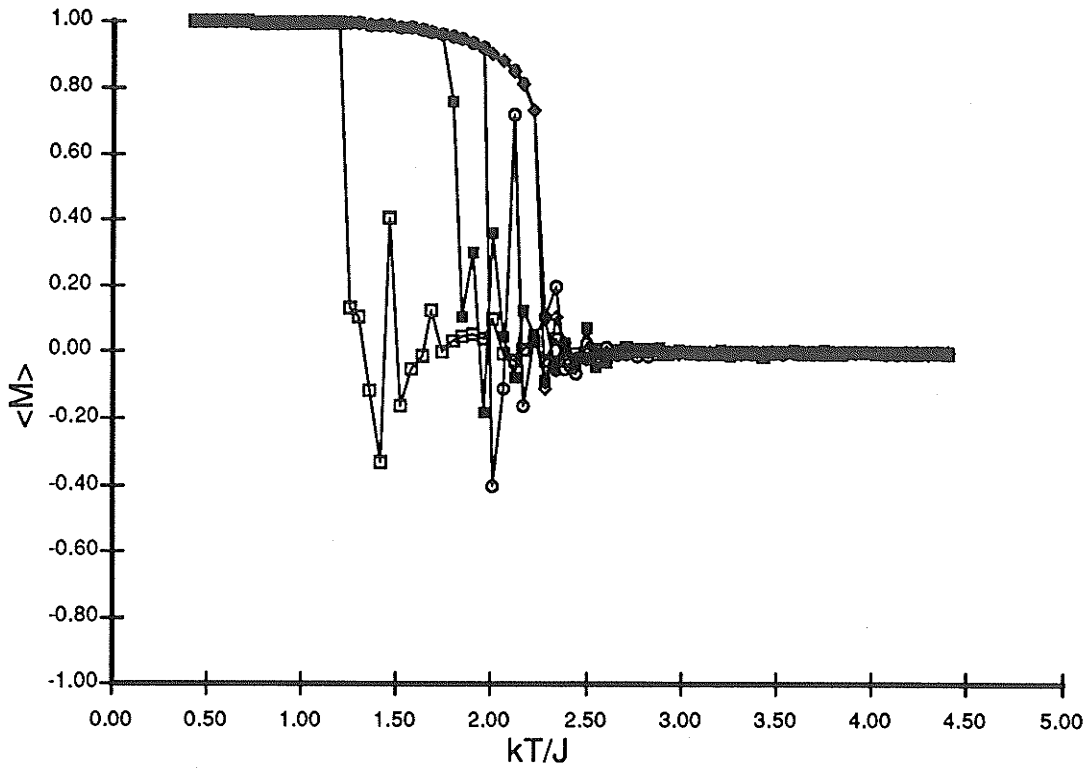


Figure 3.24 : *<M> versus kT/J for various lattice sizes. Notice the overturning of the lattice especially for small lattice sizes.*

The corresponding probability of flipping the spin, $e^{\Delta E/kT}$ is then compared to a random number, x, uniformly distributed between 0 and 1. If $x < e^{\Delta E/kT}$ then the spin is flipped, otherwise it is left alone [Metropolis1953]. This method has been employed by many workers, for example [Pawley1984], [Binder1980], [Landau1980], [Stoll1973], [Landau1976b] and in some cases special-purpose processors incorporating pipelining have been constructed for Ising model calculations [Pearson1983a], [Barber1985], [Hoogland1983].

As with the percolation model critical exponents are used to describe the behaviour of the Ising model at the critical point. The method of analysis will be via computer simulation using the Monte Carlo method described above. Here we will again restrict our attention to the two-dimensional Ising model, even though much is known analytically about its physical properties [Onsager1944], [McCoy1973]. We note that when higher dimensions are considered, Monte Carlo simulation of the lattice is generally the only way of calculating the critical exponents.

3.3.1. Finding the Critical Exponents

First we must establish the critical point. For the Ising model this corresponds to the Curie temperature of the ferromagnet under study or the temperature at which the magnetisation of the ferromagnet undergoes a phase transition from ferro- to

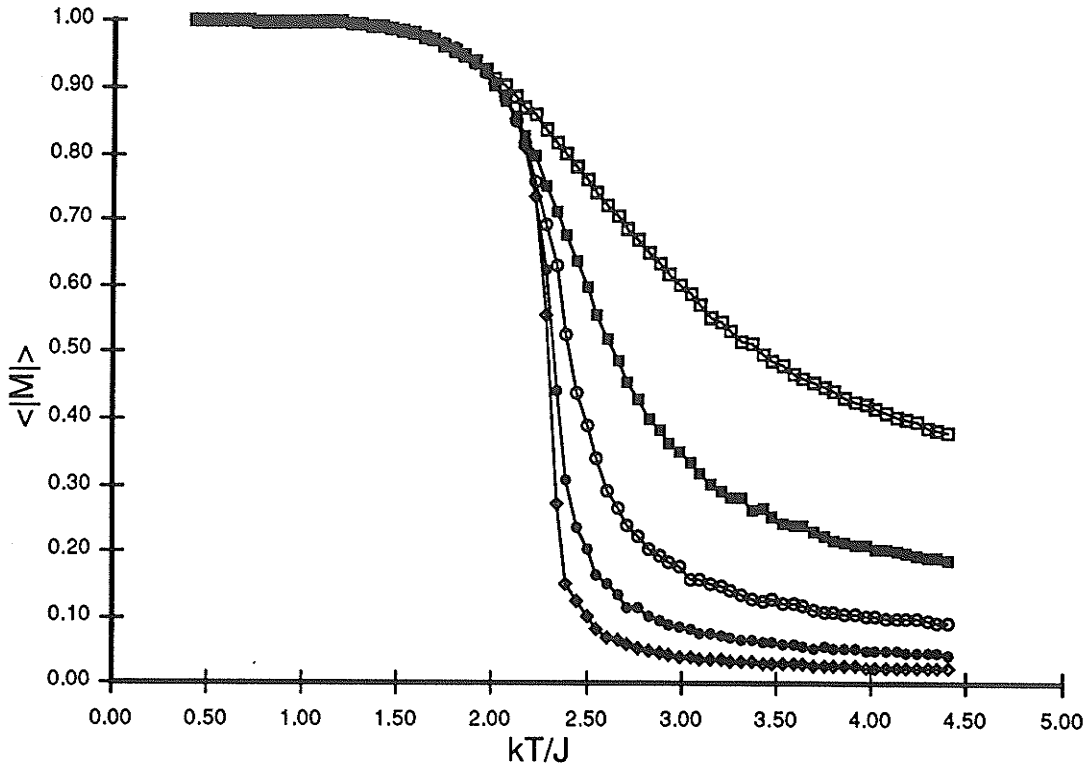


Figure 3.25 : *$<|M|>$ versus kT/J for various lattice sizes.*

paramagnetic such as that shown in Fig 3.19 for a 64 × 64 square lattice. It is known for infinite two-dimensional square lattices that $kT_c(\infty)/J = 2.269$. However, for lattices of size L, $kT_c(L)/J$ may be substantially different. All results shown in this section are from actual Monte Carlo simulations of the Ising model acting on various size square lattices. As before, results will first be shown for simulations using the SUN3-160 PRNG; later results will be shown which are derived from the proposed Ising model architectures.

For the Ising model there are generally five quantities and their associated critical exponents of interest. Here we will consider four of these quantities. The expected magnetisation and total system energy were discussed above. Two other additional thermodynamic quantities, specific heat and susceptibility are also generally extracted from the Ising model. The specific heat, C, is defined as

$$C = \frac{\partial E}{\partial T} \tag{3.11}$$

and the susceptibility, χ as

$$\chi = \frac{\partial M}{\partial H} \tag{3.12}$$

where H is a uniform magnetic field. These two quantities can be calculated based on fluctuations in the energy and the spontaneous magnetisation respectively as

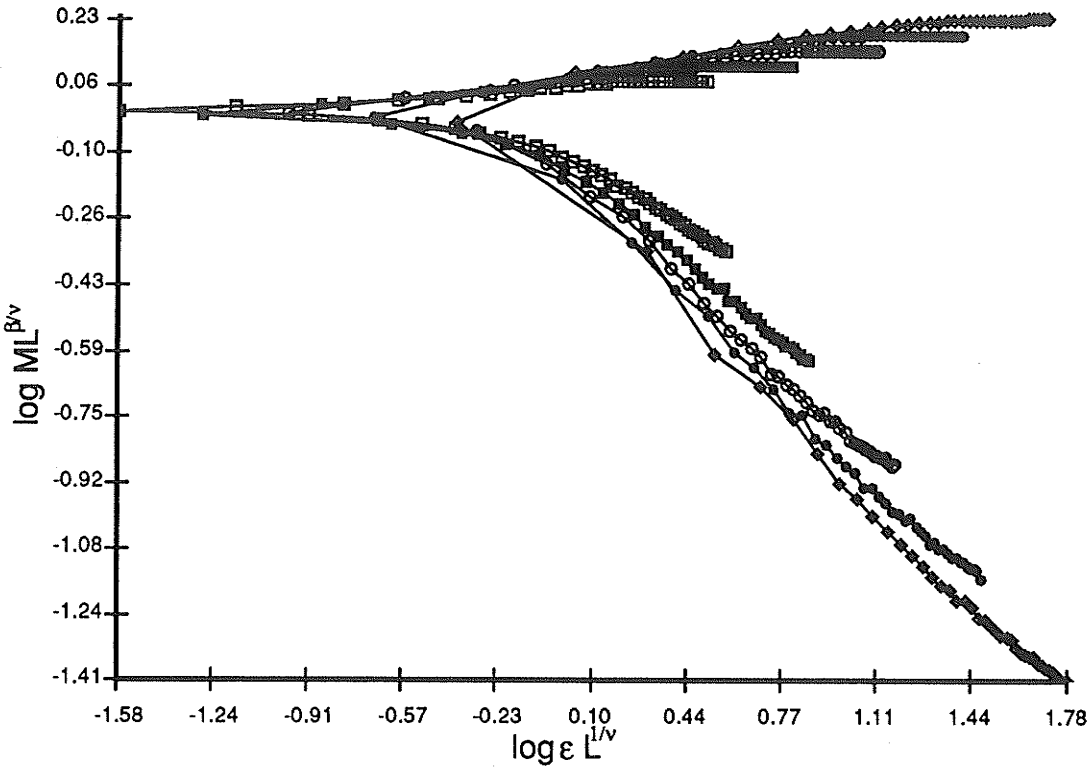


Figure 3.26 : *Finite-size scaling plot of the magnetisation with periodic boundary conditions. We define ε as in Fig. 3.23. Data points for $T < T_c(\infty)$ (right hand curves) line up with slope 0.125 and for $T > T_c(\infty)$ (left hand curves) line up with slope -0.875.*

[Landau1968]

$$C = \frac{N^2}{kT^2}\left(\langle E^2 \rangle - \langle E \rangle^2 \right) = \frac{\partial E}{\partial T} \tag{3.13}$$

$$\chi = \frac{N^2}{kT}\left(\langle M^2 \rangle - \langle M \rangle^2 \right) = \frac{\partial M}{\partial H} \tag{3.14}$$

The Monte Carlo simulation implemented here closely follows that of [Landau1976a] and [Binder1984]. Both periodic and free boundary conditions are possible. The simulation runs from low temperature, $kT/J \ll kT_c/J$ to high temperature $kT/J \gg kT_c/J$. At each temperature the lattice is initialised to all up spins and then 100 complete lattice updates are used to bring the lattice to equilibrium for temperature, T.[3.7] Magnetisation and energy measurements are then made for every complete

[3.7] Far from the critical point the number of lattice iterations to equilibrium is quite low but near the critical point the number of steps required can rise by over three orders of magnitude [Swendsen1983]. It is possible to use time-dependent correlation functions as a means of measuring the distance from equilibrium [Swendsen1983] but 100 steps to equilibrium was considered adequate for the purposes of the data to be presented here.

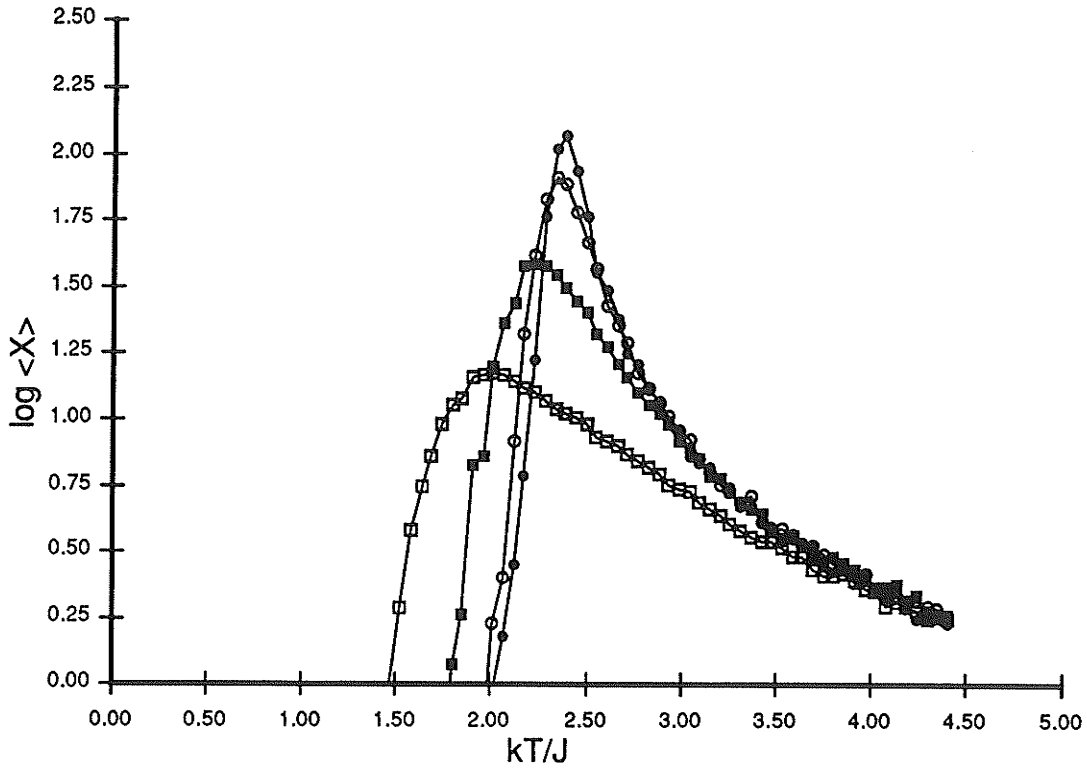


Figure 3.27 : *Susceptibility versus kT/J for various lattice sizes.*

lattice update. Note that for the specific heat and susceptibility measurements we require independent values of M and E in the calculations of $<M^2> - <M>^2$ and $<E^2> - <E>^2$ respectively. Therefore, it is best to reinitialise the lattice and bring the lattice back to equilibrium before each measurement if the specific heat or susceptibility is to be derived.

The normalised energy for both periodic and free boundary conditions is shown in Fig. 3.20. Notice that the behaviour of the small 4×4 lattice is significantly different than that of the other lattices. This shows the dramatic effect that finite size can have on Ising model behaviour. Also we see the effect of boundary conditions on the Ising model, especially for the smaller lattice sizes. The differences in behaviour between the two boundary conditions is due to the edge discontinuity affecting the energy of the edge spins.

The specific heat for lattices with periodic boundary conditions is shown in Fig. 3.21. Again we see the dramatic effect of finite lattice size on the Ising model. For the infinite lattice the specific heat at $kT_c(\infty)/J$ is infinite. However, for the finite lattices simulated here we see that the maximum value increases as the lattice size increases and approaches $kT_c(\infty)/J$ from the right. Exact curves can be found in [Ferdinand1969]. We can extrapolate $kT_c(\infty)/J$ by plotting the temperature of maximum specific heat versus L^{-1} as in Fig. 3.22. This yields an estimated $kT_c(\infty)/J$ of 2.32 as opposed to the exact value of 2.269. The behaviour of the specific heat can be

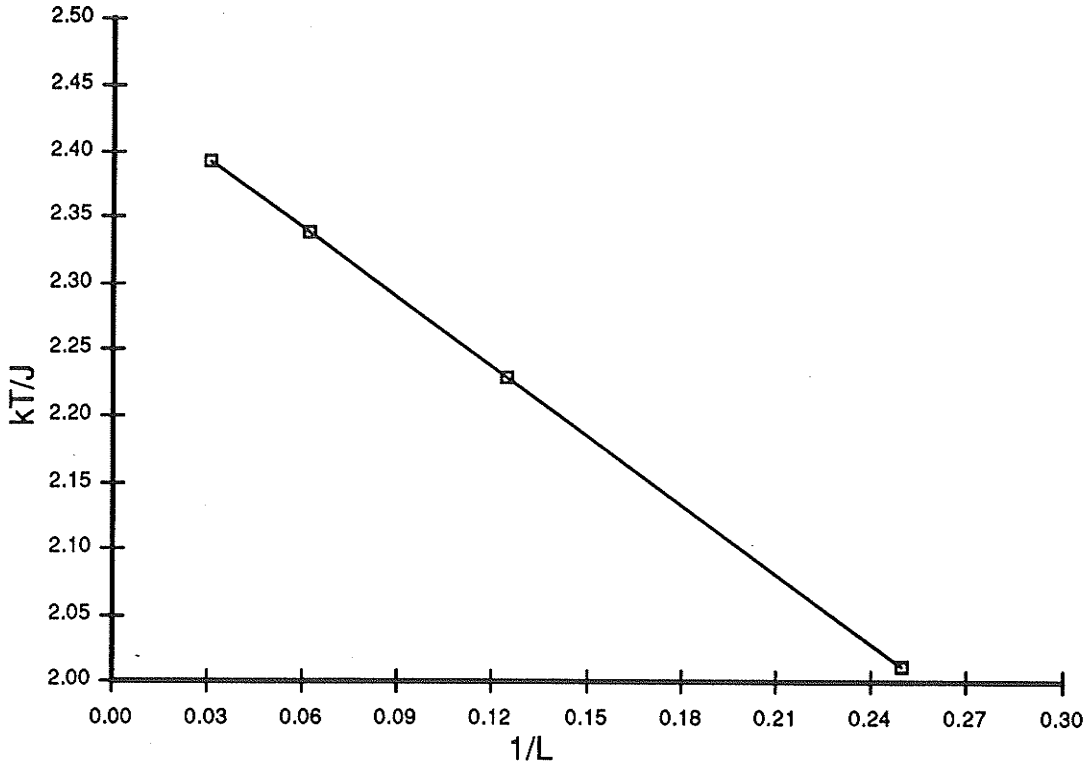


Figure 3.28 : *Extraction of $kT_c(\infty)/J$ using the susceptibility. The data points give the temperature for the maximum susceptibility.*

described by the scaling relation

$$C = L^{\alpha/\nu} Z^0(\varepsilon L^{1/\nu}) \tag{3.15}$$

where Z^0 is solely a scaling function of $\varepsilon L^{1/\nu}$,

$$\varepsilon = \frac{T - T_c(\infty)}{T_c(\infty)} \tag{3.16}$$

and α and ν are exponents for the infinite square lattice. The value of ν is known to be 1.0 for two-dimensional lattices. We can test this relation by plotting $CL^{-\alpha/\nu}$ versus $\varepsilon L^{1/\nu}$ as in Fig 3.23. Notice that the data lie on two curves one for $T < T_c(\infty)$ and the other for $T > T_c(\infty)$. For large L and therefore large $\varepsilon L^{1/\nu}$ it can be shown that the infinite lattice critical behaviour is asymptotically reproduced by [Landau1976a]

$$Z^0(\varepsilon L^{1/\nu}) \approx A \cdot (\varepsilon L^{1/\nu})^{-\alpha} \tag{3.17}$$

where A is the critical amplitude for the infinite lattice specific heat. Thus, we can draw a straight line through the data for $T < T_c(\infty)$ and $T > T_c(\infty)$ in Fig. 3.23 at large $\varepsilon L^{1/\nu}$ and describe asymptotically the behaviour of the infinite lattice. We see from the slope of this line that $\alpha = 1.0$.

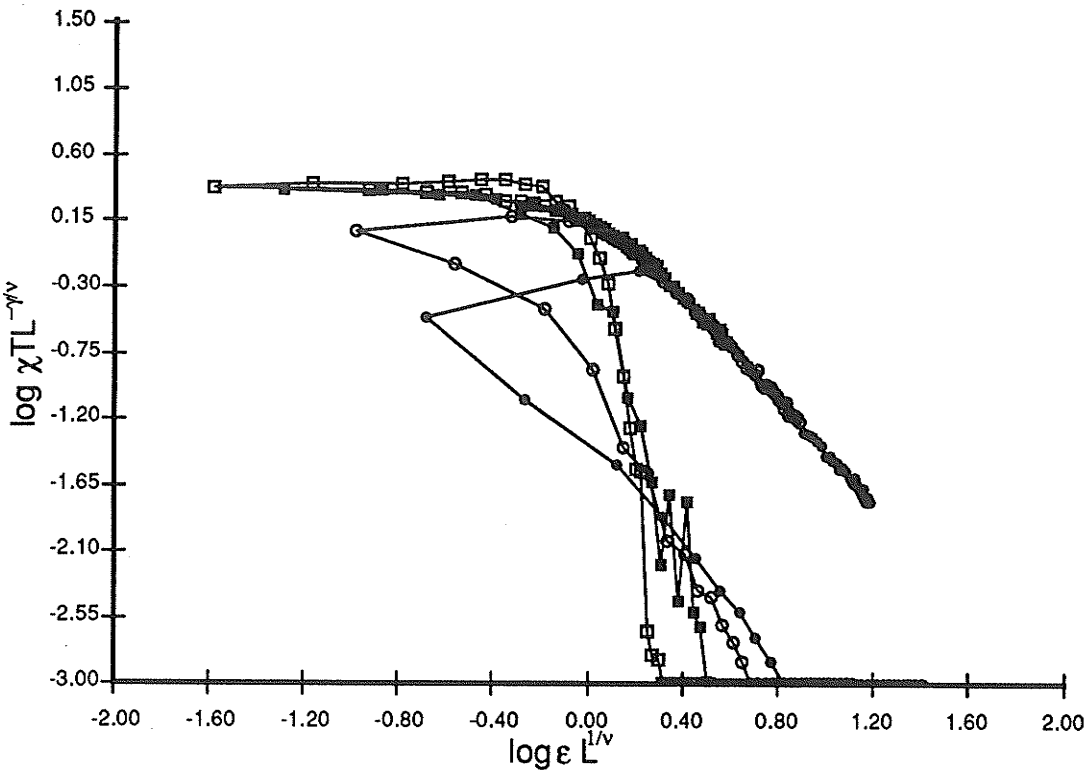


Figure 3.29 : *Finite-size scaling plot of the susceptibility of lattices with periodic boundary conditions. Here we plot $\chi TL^{-\alpha/\nu}$ versus $\varepsilon L^{1/\nu}$, with ε defined as in Fig. 3.23 Data points for $T > T_c(\infty)$ (right hand curves) and $T < T_c(\infty)$ (left hand curves) line up with slope -1.75.*

Due to the finite lattice size the entire lattice may completely overturn magnetically during the course of a series of measurements [Binder1975] as we see in Fig. 3.24. Thus, we will plot <|M|> rather than <M>. This is shown in Fig. 3.25. Note that this removes the polarity of the magnetisation. The magnetisation can be described by the scaling relation

$$M = L^{-\beta/\nu} X^0(\varepsilon L^{1/\nu}) \tag{3.18}$$

where X^0 is solely a scaling function of $\varepsilon L^{1/\nu}$. Therefore, as for the specific heat we can plot $ML^{\beta/\nu}$ versus $\varepsilon L^{1/\nu}$ to extract the value of β for infinite lattice behaviour. Here the infinite lattice magnetisation is asymptotically described by

$$X^0(\varepsilon L^{1/\nu}) \approx B \cdot (\varepsilon L^{1/\nu})^{\beta} \tag{3.19}$$

This is shown in Fig. 3.26, where we see that $\beta = 0.125$, for $T < T_c(\infty)$ and $\beta = -0.875$ for $T > T_c(\infty)$.

Susceptibility curves for various size lattices are shown in Fig. 3.27. As for the specific heat the magnitude and position of the maximum value are dependent on the lattice size. However, an additional source of error is introduced because of the

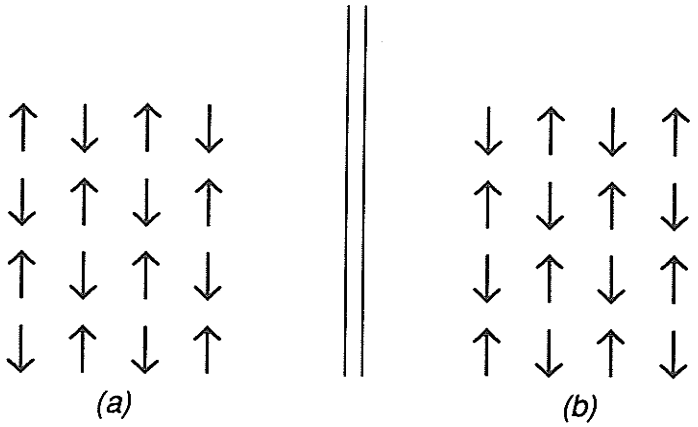


Figure 3.30 : *(a) A 4 × 4 lattice with M = 0. (b) Updated 4 × 4 lattice.*

tendency for the entire lattice of spins to reverse during the course of the computer simulation. The energy and specific heat are unaffected by this reversal since the spins remain in parallel but, since the magnetisation is changing sign during a reversal, the susceptibility will be greatly affected. Thus, it is possible to calculate two entirely different values of χ while still having similar values of $<|M|>$. The effect is reduced as the lattice size increases. As for the specific heat, it is possible to attempt to extract $kT_c(\infty)/J$ by plotting the temperature of maximum susceptibility versus the inverse of the lattice size as shown in Fig. 3.28. This yields an estimate of $kT_c(\infty)/J = 2.45$. Finally, the scaling relation which defines the susceptibility is given by

$$\chi T = L^{\gamma/\nu} Y^0(\varepsilon L^{1/\nu}) \quad . \tag{3.20}$$

where Y^0 is solely a scaling function of $\varepsilon L^{1/\nu}$. As before we plot $\chi T L^{-\gamma/\nu}$ versus $\varepsilon L^{1/\nu}$ to determine the value of γ given the asymptotic behaviour of

$$Y^0(\varepsilon L^{1/\nu}) \approx C \cdot (\varepsilon L^{1/\nu})^{-\gamma} \quad . \tag{3.21}$$

From Fig. 3.29 we see that the value of γ is 1.75 for both $T < T_c(\infty)$ and $T > T_c(\infty)$.

3.3.2. A Proposed Ising Architecture

As with the percolation model the majority of the computational work in the Ising model is in updating the lattice. At each site in the lattice we must calculate the energy and change the spin value with the corresponding probability. This involves adding the spins of the neighbour sites (four in the case of the square lattice) and negating the resulting sum if the site spin is not parallel to the sum. It is possible to calculate the probability explicitly each time the energy is calculated but since there are only a few possible energy values it is much quicker to use a lookup table for the probability values. Finally, we must generate a random number, compare it to the probability and change the spin accordingly. Therefore, updating the lattice essentially consists of two operations: calculating the local energies and generating the random number for the probability comparison. Each of these operations is repeated over the $N = L^2$ sites of

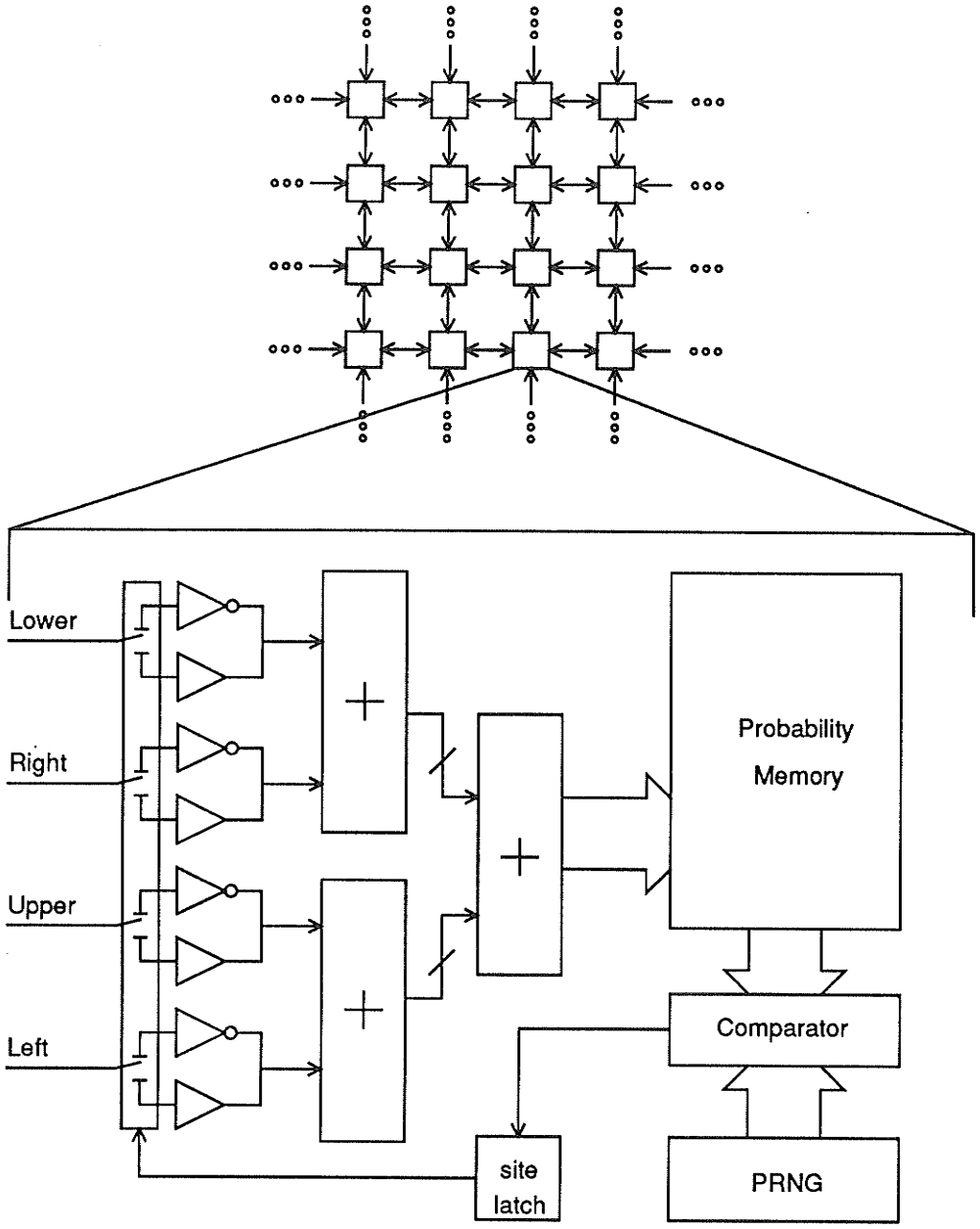


Figure 3.31 : *Proposed parallel Ising architecture using a spin processor for each site on the spin lattice.*

the lattice. One important point to note is that on odd time steps, odd numbered sites are updated and on even time steps, even numbered sites are updated. This is essential in the parallel case to avoid the so-called *feedback catastrophe* which results if all the spins are updated during the same time step [Vichniac1984]. For example, in Fig. 3.30(a) a 4×4 lattice is shown with a magnetisation of zero. If all spins are updated during the same time step then at low temperatures there is a high probability that the spin configuration of Fig. 3.30(b) will result. This is because each site has zero

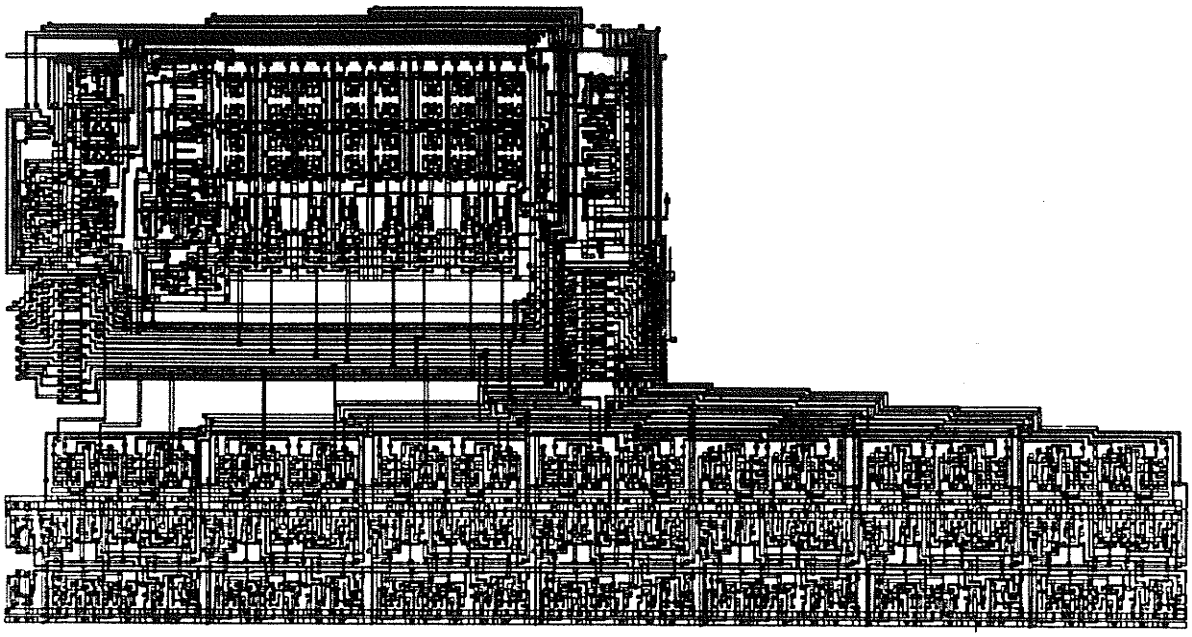


Figure 3.32 : *Layout of a 16 bit Ising spin processor.*

energy and so will flip. On the next time step the spin configuration will revert back to that of Fig. 3.30(a) for the same reason. This oscillation will continue indefinitely on succeeding time steps. The odd/even updating technique avoids this problem.

The remaining problem lies in the calculation of the critical exponents. For the same reasons as with the percolation computer, expressed here with respect to the energy and magnetisation calculations, it is argued that it is not an optimal solution to build a special purpose computer to calculate the critical exponents. Instead the greatest gains are to be realised in building a coprocessor which updates the Ising spin lattice and reports the energy and magnetisation of the resulting configuration. The actual spin configuration need be determined only if quantities such as correlation length are to be found.

Obviously tremendous speedup occurs when all the spin updating is done in parallel. As with the percolation model it is desirable to simulate as large a system as possible. The limitations on the size of Monte Carlo Ising model simulation are twofold: 1) the size of the data memory required to store the spin data and 2) the simulation time for large lattices can become prohibitively long, especially near the critical point where many iterations of the lattice are required before the spin system is at equilibrium.

To accelerate Ising model simulation the parallel architecture of Fig. 3.31 is proposed. Here each processor consists of a data memory to store the probabilities, a comparator, a PRNG, and a latch to store the spin. The PRNG is the same as that for the percolation computer. Note that it is possible to use a large bus to route the various probability values and to have a bus selector at each processor which chooses the bus corresponding to the desired probability. Here we consider a data memory

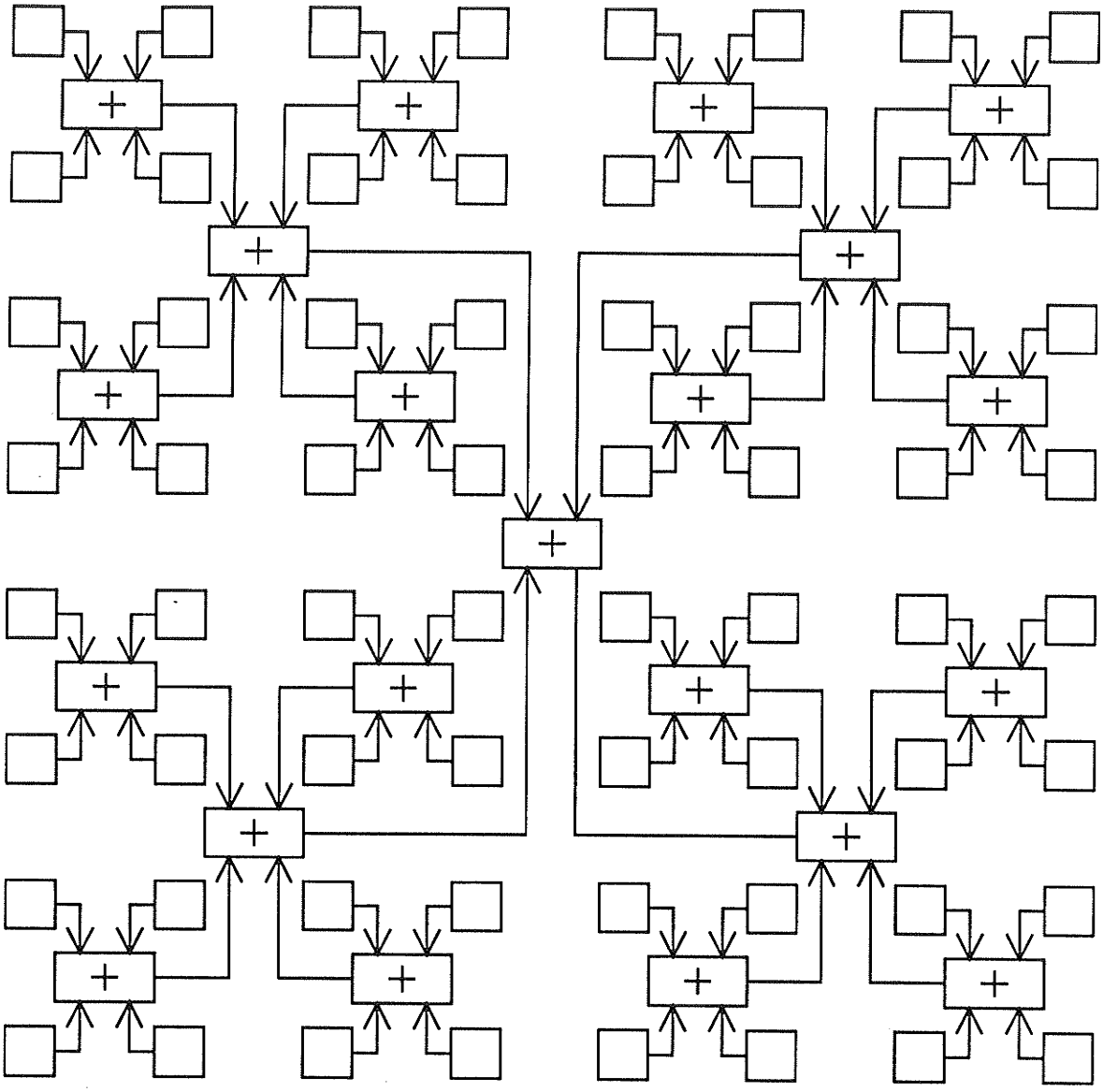


Figure 3.33 : *An adder tree with a branching ratio of four.*

containing the probabilities at each processor since it is conceptually easier to understand. Technological considerations will determine the actual configuration used in implementation. Each processing site corresponds to a single spin site and we will refer to this architecture as the L^2 spin processor architecture. The spins are stored as 0 for a down spin and 1 for an up spin. The energy is calculated as the sum of the four neighbour spins (here we consider the square lattice). If the spin is down (i.e. 0) the the neighbouring spin values are inverted as they are summed. The use of 0 and 1 as spin values forces one to consider the energy of each spin as a number in the range [0,4] where 0 corresponds to maximum energy and 4 to minimum energy. The energy is then used to address the probability memory, where each word is the probability of a spin with the energy of its address being flipped. The corresponding

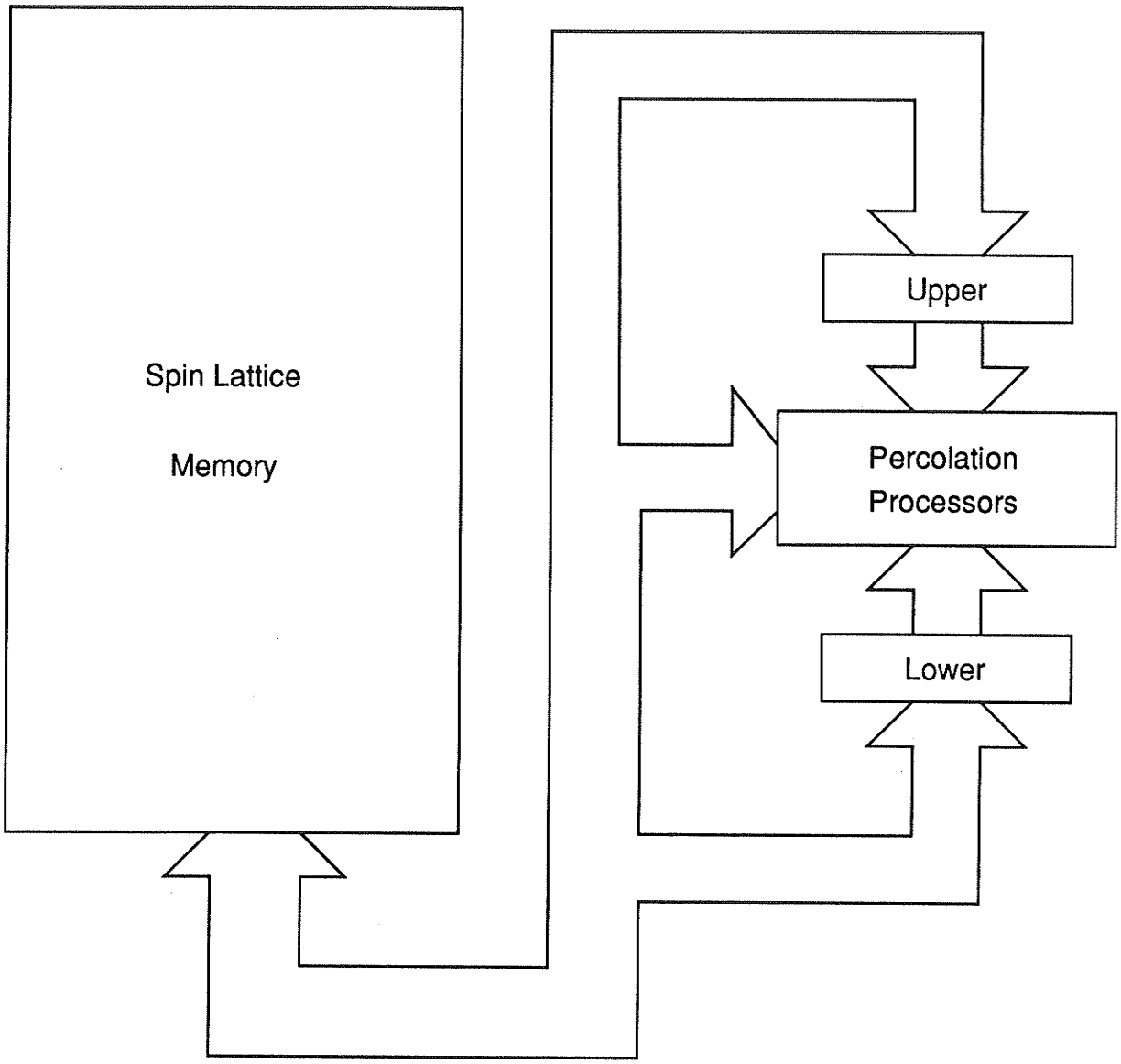


Figure 3.34 (a): *Row at a time spin updating using three row registers.*

probability is then compared to a number from the PRNG and the spin is flipped if the number from the PRNG is less than that probability value. If the updating is restricted to the even/odd scheme required to avoid the feedback catastrophe then all even/odd spins can be updated in parallel. It is possible to combine even and odd sites into one processor with two spin latches due the updating scheme. This architecture will result in a time saving of $O(N)$, where $N = L^2$, over conventional serial Monte Carlo simulation. In addition, the time for each spin update in this Ising computer is very small since it can occur in one clock cycle ($\approx$ 50nsec.) while for a general purpose computer the four additions, two decisions, and miscellaneous memory references will require many clock cycles ($\geq$ 10μsec.). The overall acceleration is approximately $200N$ as

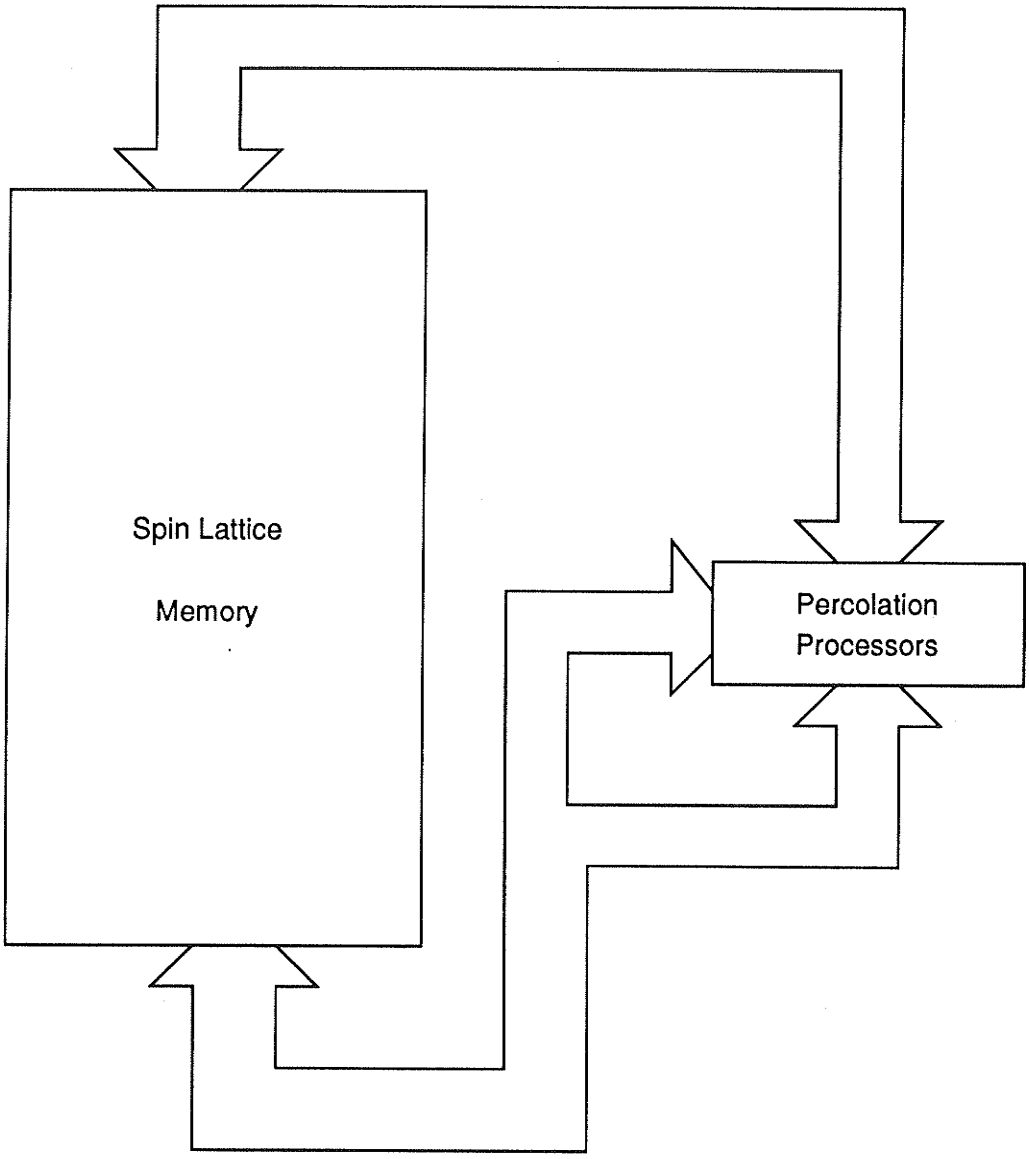


Figure 3.34 (b): *Row at a time spin updating using a two-port data memory.*

opposed to a serial computer.

The regularity of each spin processor makes the Ising computer an ideal candidate for a VLSI implementation. However, we again find that the size of the lattice which can be simulated is directly dependent on the size of the processor needed at each site. A 16 bit processor is pictured in Fig. 3.32. Using the technology described in Chapter 2 the size is 1.975 mm^2 so on a 4.8 $\times$ 4.8 mm. die it is possible to fit only ten such processors, or twenty spin sites, if the combining of even/odd site processors is used.[3.8] It is unrealistic to consider simulating lattices larger than 1000 $\times$ 1000 spins using this architecture with current technology. Temporarily the discussion will be

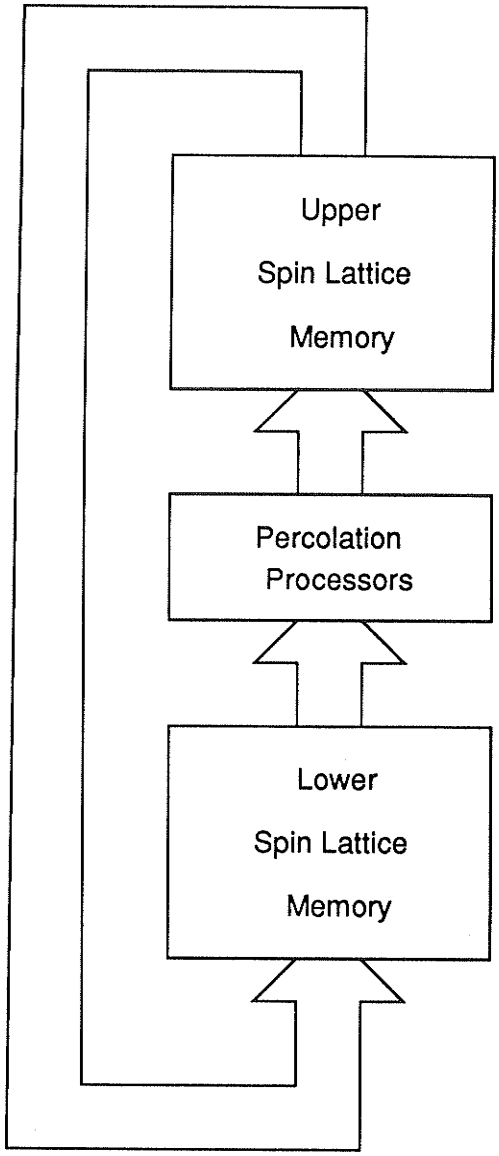


Figure 3.34 (c):*Row at a time spin updating using a shifting data memory.*

restricted to lattices of size $L \leq 1000$ but we return to the problem of larger lattices later in the chapter.

While it is possible to use the L^2 spin processor architecture to substantially increase the speed of lattice updating, further simulation speed increases could be achieved by calculating total system parameters such as the magnetisation and energy. Both these operations are inherently local with global properties found by

[3.8] As for the percolation computer it is possible have up to 1000 Ising site processors per chip with the area saving approach mentioned above and a state of the art CMOS technology.

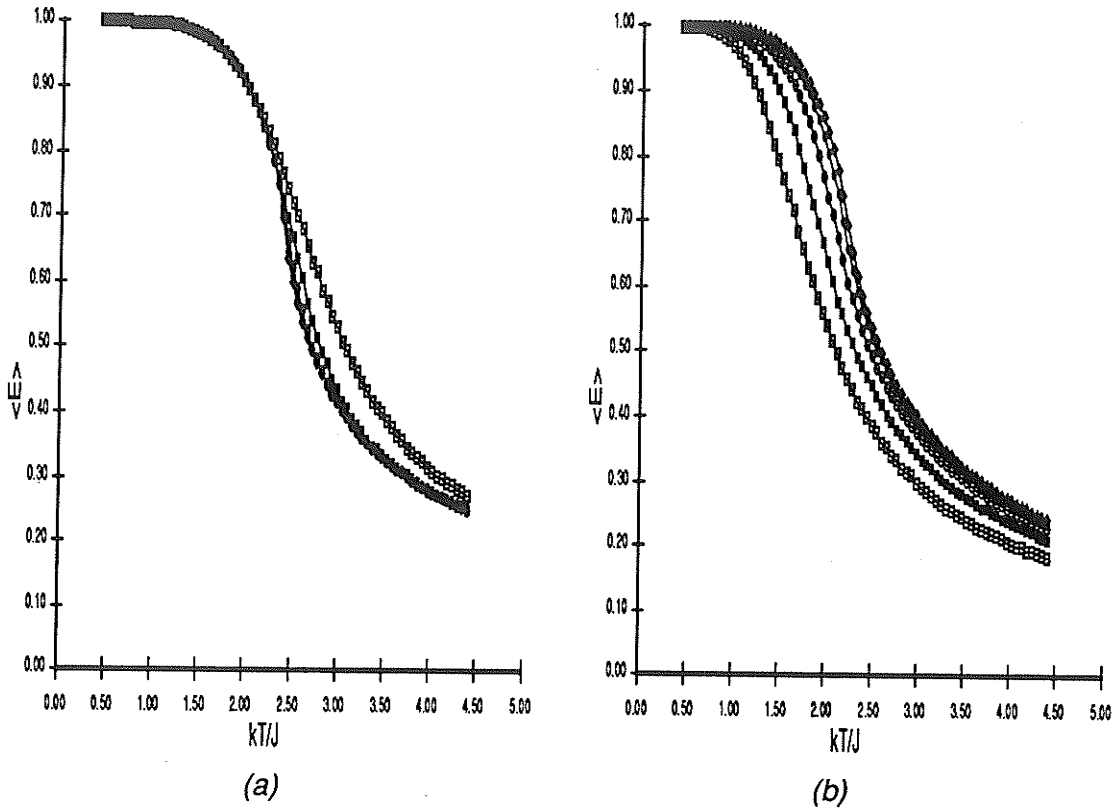


Figure 3.35 : *Normalised energy versus kT/J using the L^2 spin processor with (a) periodic boundary conditions. (b) free boundary conditions.*

simply summing the local results. We have already reviewed the calculation of the spin energy at each site in the lattice. The magnetisation at each site is simply the value of the spin. The global sum of both the local energy and magnetisation can be found by placing the adder tree of Fig. 3.33 on top of the lattice updating architecture of Fig. 3.31. Thus, we require only $O(log\ L)$ steps to calculate both the magnetisation and energy for the entire lattice. The circuit of Fig. 3.32 has been demonstrated to possess an operating speed of at least 20 MHz and the adder tree should easily operate at this clock speed so, in less than 1μsec., a 1000 × 1000 lattice can be updated and the energy and magnetisation reported to the host computer. This makes it possible to do extensive simulation of reasonably large spin systems close to the critical point, even though it often takes several thousand lattice updates to achieve equilibrium. This requires a prohibitively long time on a general purpose computer but only a few milliseconds on this Ising computer.

Calculation of quantities such as correlation length of the magnetic clusters at or near the critical point can also be greatly aided by the Ising computer. In the percolation model a technique for grouping clusters of occupied sites was developed and it is a simple matter to adapt this same clustering architecture for grouping of like spin types. The lattice configuration can then be offloaded to the host computer for final

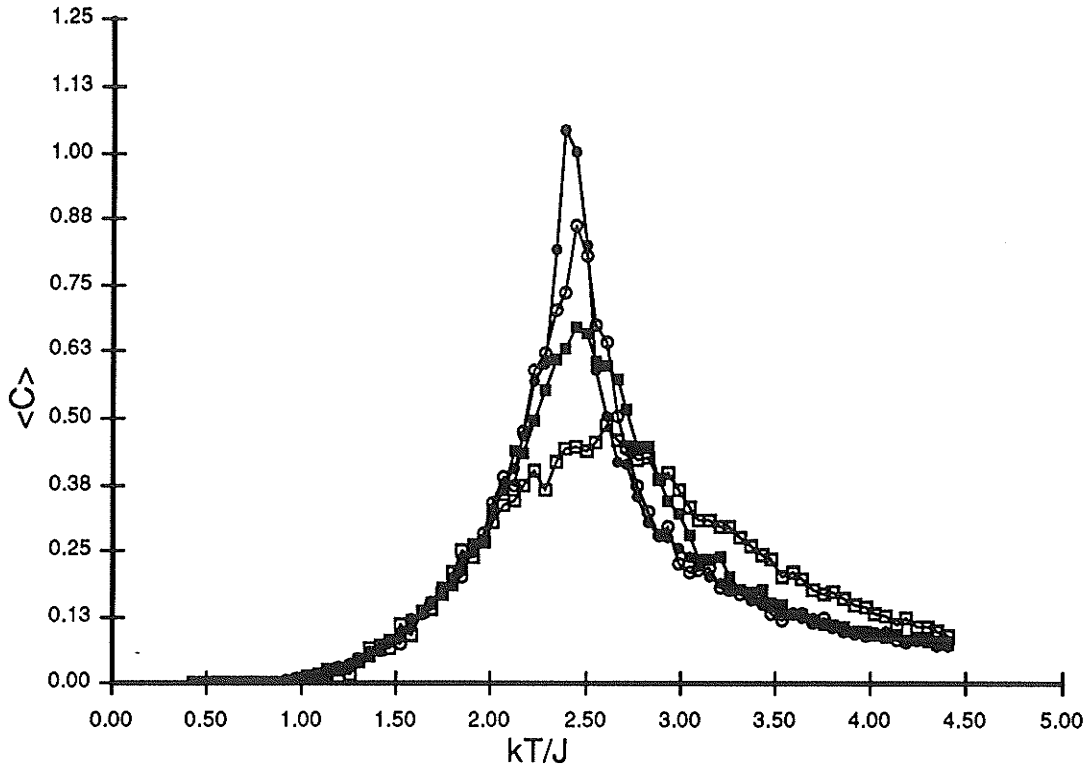


Figure 3.36 : *Specific heat versus kT/J using the L^2 spin processor architecture with periodic boundary conditions.*

processing. As before, these calculations will be greatly accelerated if like spins have already been grouped into clusters. However, the greatest speedup for these calculations lie in the fact that equilibrium (i.e. the time at which the computer experiment can actually begin) can be achieved in only a few milliseconds, whereas on general purpose computing hardware this could take many hours. In addition, many of these calculations are best made by removing the history dependence of the individual measurements. This can only be accomplished by reinitialising the lattice to a completely parallel spin configuration and waiting for the system to return to equilibrium. This is again an impossibly long and expensive procedure for conventional simulation but easily possible using the Ising computer.

Above we have only considered lattices of size $L \leq 1000$. We will now consider much larger systems. For these systems we will update the lattice one row at a time. The problem here is that we require the values of the spins in the rows above and below the current row. This can be accomplished using the architectures of Fig. 3.34. In this case we use a very large data memory to store all the lattice spins. The current row is updated by the same technique as described above, only we must ensure that the upper and lower rows are available to the spin processors. This can be accomplished in three ways. The first method, shown in Fig. 3.34(a) is to store the three rows in a register set. The upper row which has already been modified is in the top register,

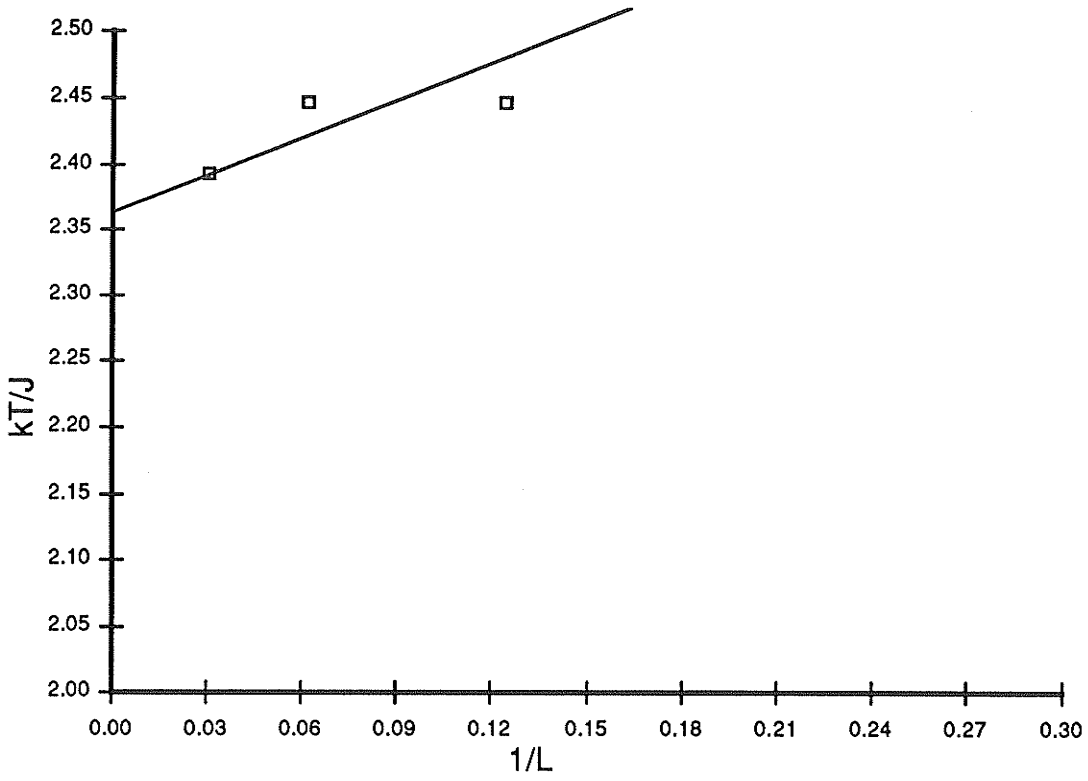


Figure 3.37 : *Extraction of $kT_c(\infty)/J$ using the the specific heat. Simulation of the L^2 spin processor architecture. The data points give the temperature of maximum specific heat.*

the current row which is about to be modified is in the middle set of registers which consist of the spin processor chain, and the lower row is loaded from the data memory into the bottom register. After the current row has been updated the top row is loaded back into the data memory, the current row is shifted to the top row, the bottom row is shifted into the spin processors, and the new bottom row is loaded from memory. This continues until all the rows of the lattice have been updated. The second method of Fig. 3.34(b) involves using a two-port memory with special addressing circuitry so as to make the upper and lower rows available to the spin processors from the spin lattice memory. Each current row must be loaded into the spin processors, updated, and the new spin values restored to memory. This does not actually save any memory transfers since for each row update we must still load and restore an entire row of spin values. An improvement on this technique would be the use of a three-port memory to make the upper, lower, and current row directly available to the spin processors, but to implement a large capacity three-port memory would be prohibitive. The final technique shown in Fig. 3.34(c) is to modify the memory so that the site processors constitute one row of the memory. Each row is then shifted up one level, after the current row has been updated. This will increase the size of the data memory but avoids the I/O problems of the first two methods. It also provides a much more regular structure

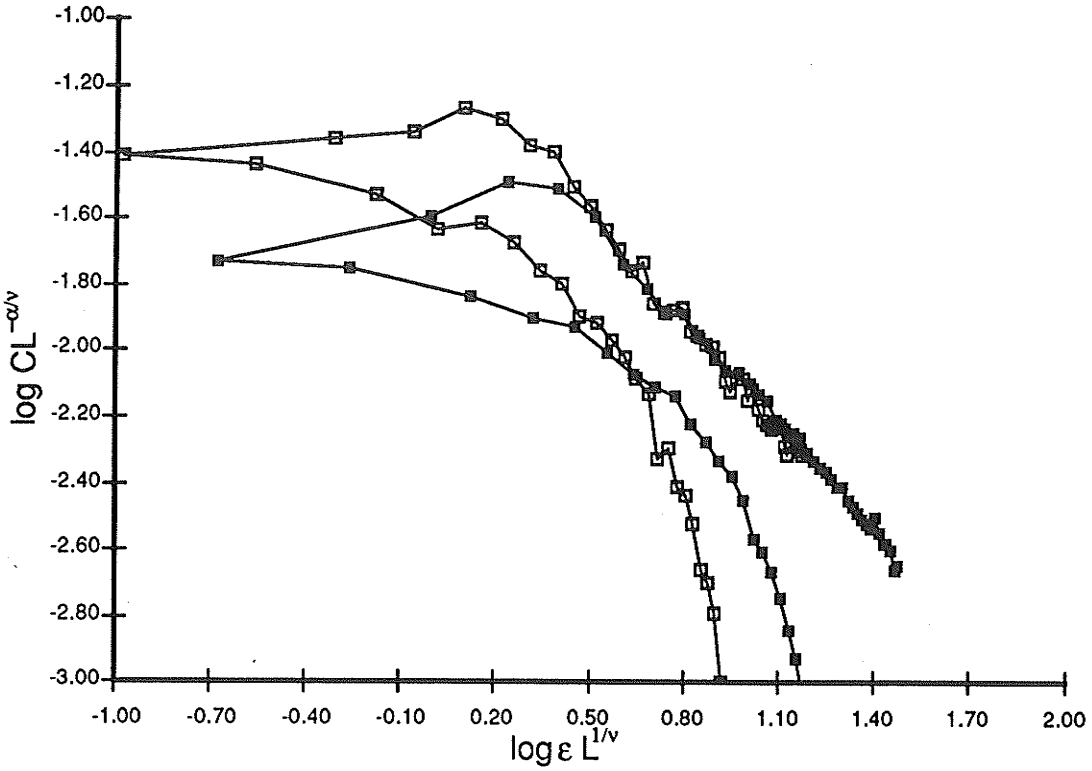


Figure 3.38 : *Finite-size scaling plot of the specific heat of lattices with periodic boundary conditions using the L^2 spin processor architecture. Here we plot $CL^{-\alpha/\nu}$ versus $\varepsilon L^{1/\nu}$. Data points for $T > T_c(\infty)$ (right hand curves) and $T < T_c(\infty)$ (left hand curves) line up with slope -1.0.*

for implementation.

To determine the energy and magnetisation of the lattice we once again use the adder tree of Fig. 3.33, except that now we use a branching ratio of two. The sum is only over one row of the lattice, so we must use another summing register to add the row sums. The row sums are added until the entire lattice has been updated and the lattice energy and magnetisation are made available to the host computer. It is possible to construct the data memory in such a way that the host computer can access it to perform other calculations such as correlation length.

3.3.3. Simulation Results for the L^2 Spin Processor Ising Computer

Simulations of the proposed L^2 spin processor Ising computer were carried out using the same analysis techniques described previously and are now reported.

The normalised energy for both periodic and free boundary conditions are shown in Fig. 3.35. As for the previous simulations we see the dramatic effect of the lattice size on Ising model behaviour. Specific heat is shown in Fig. 3.36. The extraction of the value of $kT_c(\infty)/J = 2.36$ is shown in Fig. 3.37. Using the scaling relations

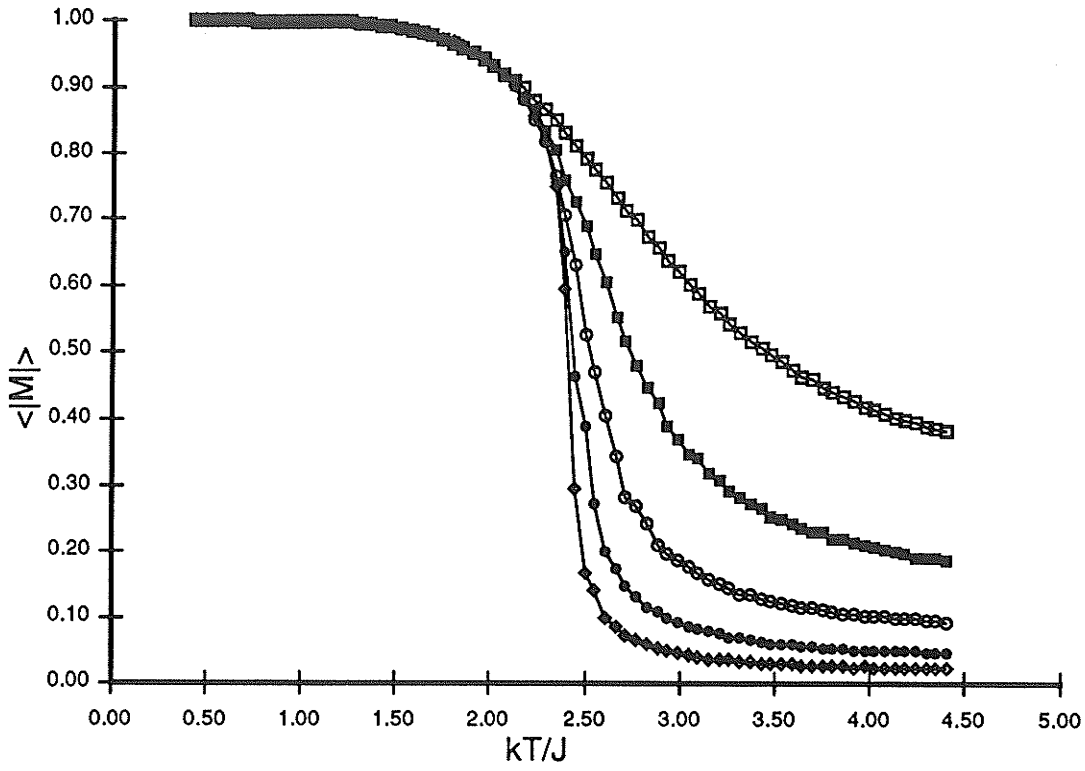


Figure 3.39 : *<|M|> versus kT/J for various lattice sizes using the L^2 spin processor architecture.*

discussed previously we see in Fig. 3.38 that the value of the critical exponent α is again 1.0. In Fig. 3.39 we plot <|M|> versus temperature and extract the value of critical exponent β from Fig. 3.40 to be 0.125 for $T < T_c(\infty)$ and -0.875 for $T > T_c(\infty)$. Susceptibility is shown in Fig. 3.41 from which we can derive Figs. 3.42 and 3.43, yielding an estimate for $kT_c(\infty)/J$ of 2.57 and $\gamma = 1.75$.

We can see close agreement between the exact values of the critical exponents, the values using the SUN PRNG, and the results derived by simulation of the L^2 spin processor Ising computer. A final check on the results derived from the Ising computer is to compare the values of <E> and <|M|> for various values of kT/J versus the exact results for the two-dimensional lattice [Onsager1944]. These results are reported in Table 3.4 from which we can see that the results derived from the proposed Ising computer correspond within their error to the exact results and also to the standard Monte Carlo simulation technique. The error can be reduced by using a larger number of samples. Therefore, we can conclude that the L^2 spin processor Ising computer will properly simulate the Ising model.

Site spacing effects on the CA rule 30 based parallel PRNG were not tested since the correct results were produced without site spacing. However, it should be noted that if the parallel LFSR is used as the PRNG the results are demonstrably poor.

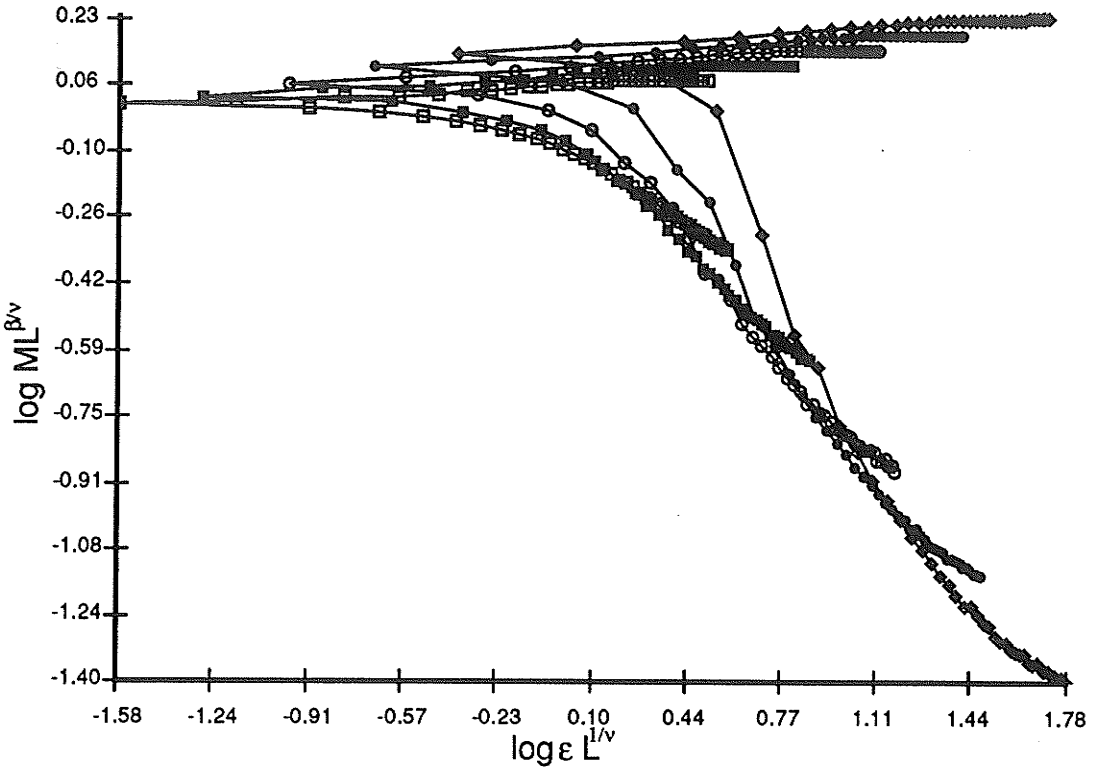


Figure 3.40 : *Finite-size scaling plot of the magnetisation with periodic boundary conditions using the L^2 spin processor architecture. We define ε as in Fig. 3.23. Data points for $T < T_c(\infty)$ (right hand curves) line up with slope 0.125 and for $T > T_c(\infty)$ (left hand curves) line up with slope -0.875.*

Therefore, we can say that the properties inherent in the CA rule 30 based PRNG are suitable for simulation of the Ising model.

Merely simulating the two-dimensional square Ising model is somewhat pointless since it has been solved analytically. However, it is a simple matter to increase the connectivity at each spin processor to correspond to that of other two-dimensional lattices such as the triangular and hexagonal lattices. In addition, three-dimensional lattices can also be simulated by changing the connectivity of the spin processors.

3.3.4. Using Renormalisation on the L^2 Spin Processor Ising Computer

We consider the application of renormalisation group techniques to the Ising model. The renormalisation rule which we will consider here is the replacement of a block of spins by one spin which represents the gross behaviour of the block of spins. For example, we could use a majority rule to replace a block of 2 × 2 spins by one spin (ties would be decided by a random number). This would progressively reduce the edge size of the lattice under consideration by a factor 2 for each renormalisation.

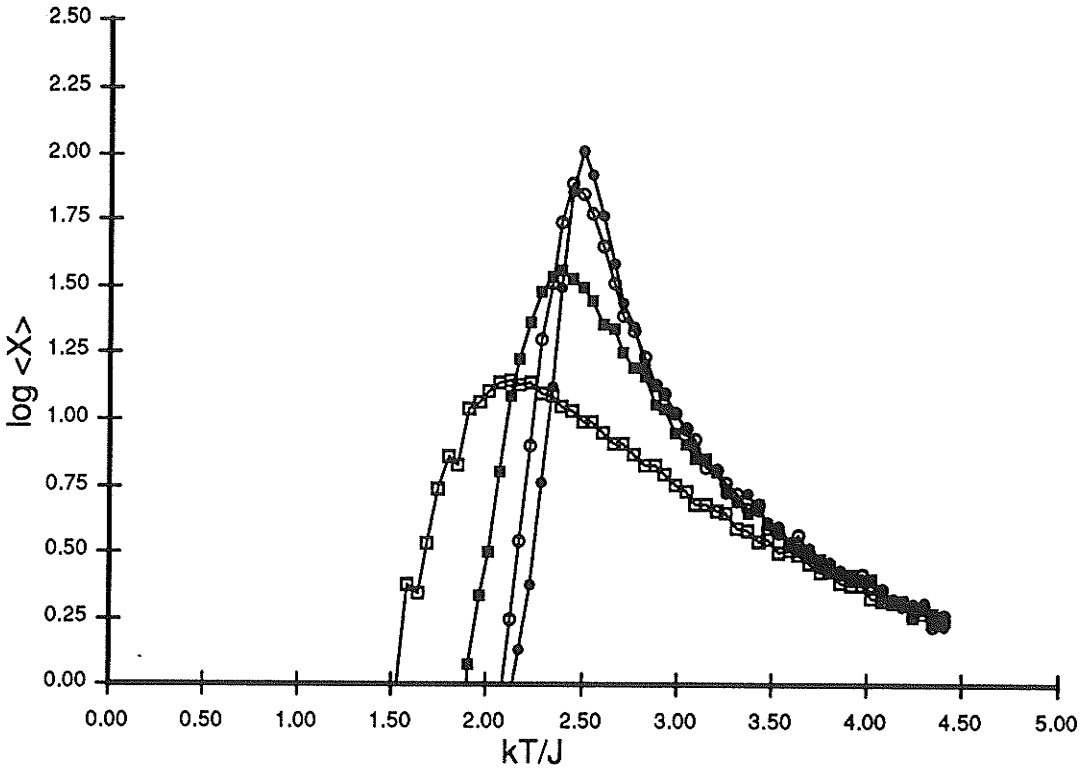


Figure 3.41 : *Susceptibility versus kT/J for various lattice sizes using the L^2 spin processor architecture.*

As with percolation, it is doubtful whether application of the renormalisation group is appropriate in a parallel architecture such as that described here. This is due to the fact that if we are analysing quantities such as the lattice energy and the magnetisation these quantities are already directly available. In addition, to reduce the effective size of the lattice by renormalisation requires as much time as the actual calculation of lattice energy and magnetisation. Finally, we must also consider the problem of examining the renormalised lattices. This can only be done by offloading each renormalised lattice, including the original lattice, from the Ising computer, a process requiring at least $O(L^2)$ time. This will reduce the computational speed of this aspect of the Ising model simulation to that of conventional serial processing.[3.9] Thus, it would appear that renormalisation is not necessarily a useful feature to build into a parallel Ising model simulator since it will significantly degrade the performance of the simulation. However, this ignores some of the other benefits which result from application of the renormalisation group. Firstly, the renormalisation technique allows very accurate

[3.9] It is possible that if enough I/O pins are available to each chip that the $O(L^2)$ factor could be reduced significantly. However, this will still cause significant throughput problems since the data collection from the Ising model simulation is bounded by an inherently serial process.

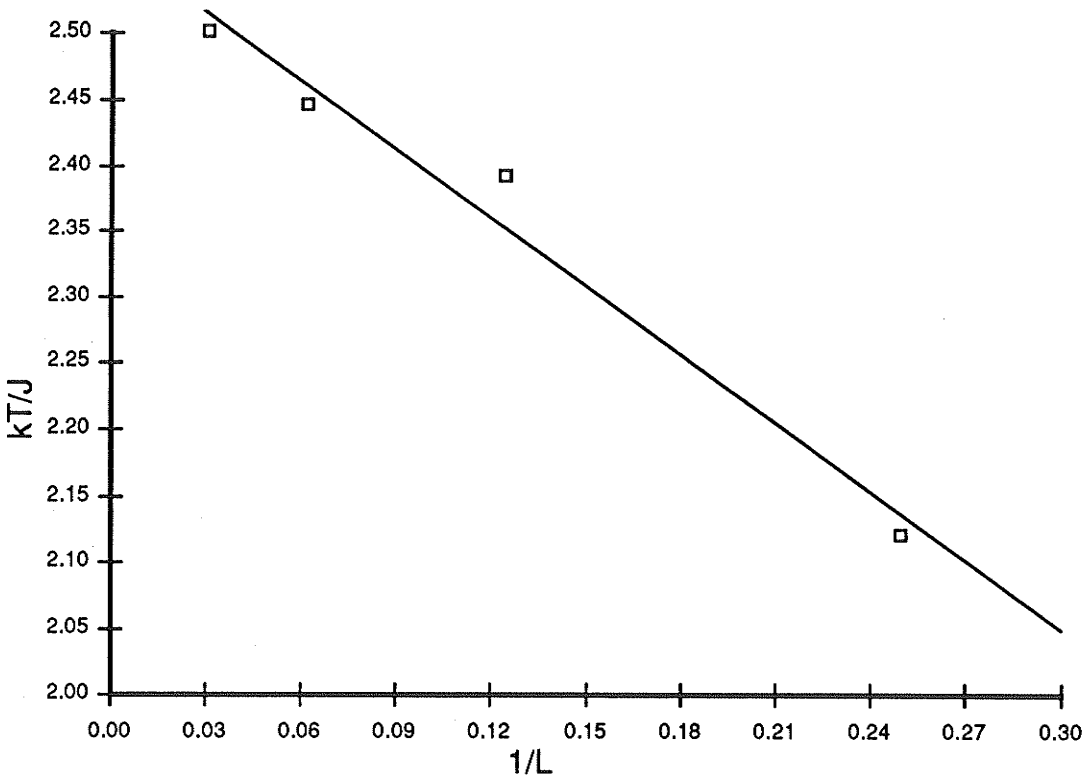


Figure 3.42 : *Extraction of $kT_c(\infty)/J$ using the susceptibility. Simulation of the L^2 spin processor architecture. The data points give temperature of maximum susceptibility.*

determination of the critical point. In addition, it provides a mechanism for studying correlation length at the critical point. A final benefit is that it will allow the host computer to study the actual lattice. This does not imply that after each update the entire lattice will be offloaded but rather that it is possible to look at the lattice if curious behaviour is present (i.e. a black box that can be opened is better than a black box which cannot). Offloading the lattice will also make it easier to test the Ising computer for electrical faults. Consider examing a black box which is supplying results from a Monte Carlo simulation of a process which is not completely understood; how do you tell if the box is broken?

A 2×2 majority rule renormalisation approach can be easily incorporated into the Ising computer by simply adding an extra storage element, the renormalisation latch, and a comparator to each spin processor as shown in Fig. 3.44. The majority rule is implemented by using the bottom level of the magnetisation adder tree. The sum of four lattice spins are found at the lowest level of this tree, if the sum is less than 2 then the replacement spin must be down, if the sum is greater than 2 then the replacement spin is up, otherwise the sum equals 2 and the replacement spin is assigned randomly based on the value of the least significant bit of the site processor's PRNG. The replacement spin for spins (x,y), $(x+1,y)$, $(x+1,y+1)$, $(x,y+1)$ is stored in the

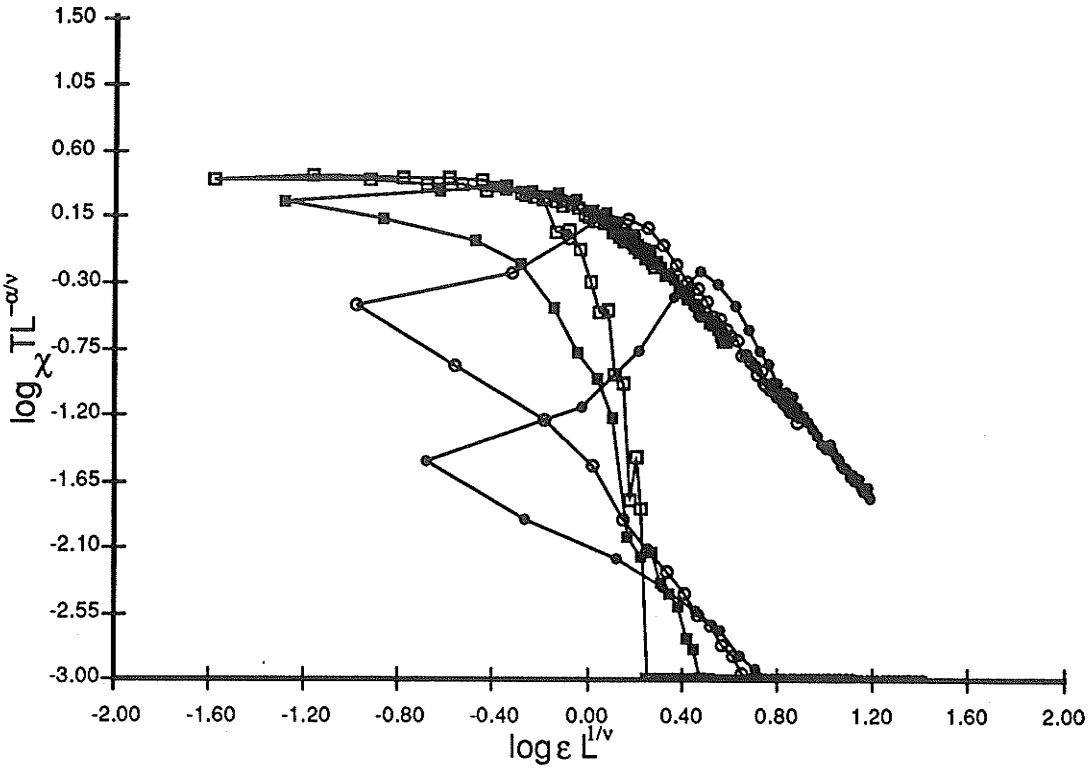


Figure 3.43 : *Finite-size scaling plot of the susceptibility of lattices with periodic boundary conditions using the L^2 spin processor architecture. Here we plot $\chi TL^{-\alpha/\nu}$ versus $\varepsilon L^{1/\nu}$, with ε defined as in Fig. 3.23 Data points for $T > T_c(\infty)$ (right hand curves) and $T < T_c(\infty)$ (left hand curves) line up with slope -1.75.*

renormalisation latch at spin ($x/2$, $y/2$). This process can be repeated indefinitely, if desired, until there is only one spin left. The renormalisation latch at each site processor is required since we must not lose the original lattice configuration. In the normal operating mode the spin is stored in both the spin and renormalisation latches and the local sums for both the energy and magnetisation are now made available from the renormalisation latch to allow energy and magnetisation calculations of the renormalised lattice. The control circuitry at each spin processor must now ensure that the first step in the lattice update process involves transferring the value in the spin latch to the renormalisation latch.

Offloading of the spin lattice may be accomplished in two ways. If the number of pins is small then the spins must be offloaded row by row, one spin at a time. This can be done by incorporating a long shift chain through the renormalisation latches of each spin processor row and shifting the bits from each row out serially. Each row would then be sequentially connected to the lattice output pin. If the number of pins is sufficient, it is possible to offload the spin lattice a column at a time using the same arrangement, only here each row is connected to a unique pin. The difference in

	Exact	Proposed		Standard	
kT/J	$<E_N>$	$<E_N>$	error	$<E_N>$	error
1.087	0.99724	0.997631	0.000733	0.996979	0.000881
1.449	0.98006	0.983695	0.006937	0.980650	0.007444
1.811	0.92693	0.940769	0.034170	0.928062	0.043008
2.173	0.81921	0.852129	0.111073	0.819579	0.112802
2.536	0.67508	0.711850	0.195514	0.675657	0.183635
2.898	0.54069	0.562358	0.213332	0.541095	0.172538
3.260	0.43873	0.443605	0.161029	0.441105	0.135338
3.622	0.36635	0.370115	0.115244	0.364889	0.104584

	Exact	Proposed		Standard	
kT/J	$<\|M\|>$	$<\|M\|>$	error	$<\|M\|>$	error
1.087	0.99859	0.998782	0.000200	0.998442	0.000250
1.449	0.98898	0.991230	0.002472	0.989410	0.002791
1.811	0.95465	0.964309	0.017988	0.955280	0.026655
2.173	0.87345	0.899275	0.080682	0.873977	0.087653
2.536	0.75207	0.784842	0.168579	0.751703	0.172171
2.898	0.63103	0.651159	0.209552	0.631363	0.183713
3.260	0.53605	0.540927	0.176900	0.537914	0.157969
3.622	0.46804	0.472880	0.141470	0.466296	0.131081

Table 3.4: *Comparison of the Monte Carlo data from the L^2 spin processor architecture with a standard Monte Carlo simulation and exact analytical results. The Ising model simulated was a 4×4 square lattice with periodic boundary conditions.*

speed between the two arrangements is $O(L)$ which can be significant for large lattices. A final consideration is that it may be possible to offload all the spins simultaneously in one step if the number of available pins equals the number of spin sites. However, this is technologically naive for large lattice sizes and we must also consider at what data rate the host computer could accept the spin lattice.

3.3.5. Another Approach

The foundations for this approach to an Ising computer arise from two recent observations: (i) the correspondence between the time evolution of d-dimensional stochastic cellular automata and the equilibrium statistical mechanics of $(d+1)$-dimensional Ising models [Verhangen1976], [Enting1977], [Enting1978], [Domany1984] and (ii) the discovery by several authors that the behaviour of recursive nonlinear systems such as one-dimensional cellular automata of certain classes exhibit effectively

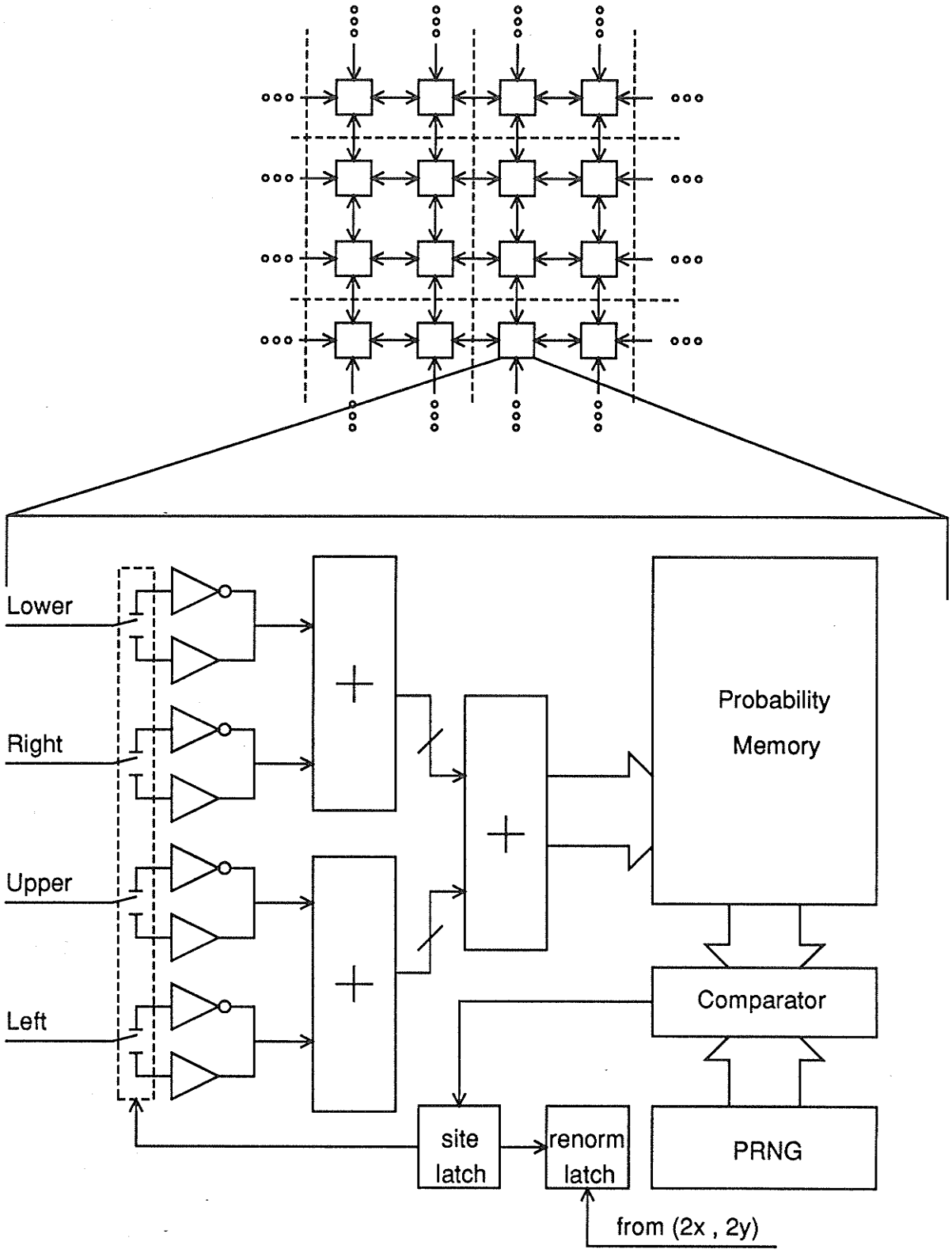


Figure 3.44 : *Modified L^2 site processor architecture to handle renormalisation.*

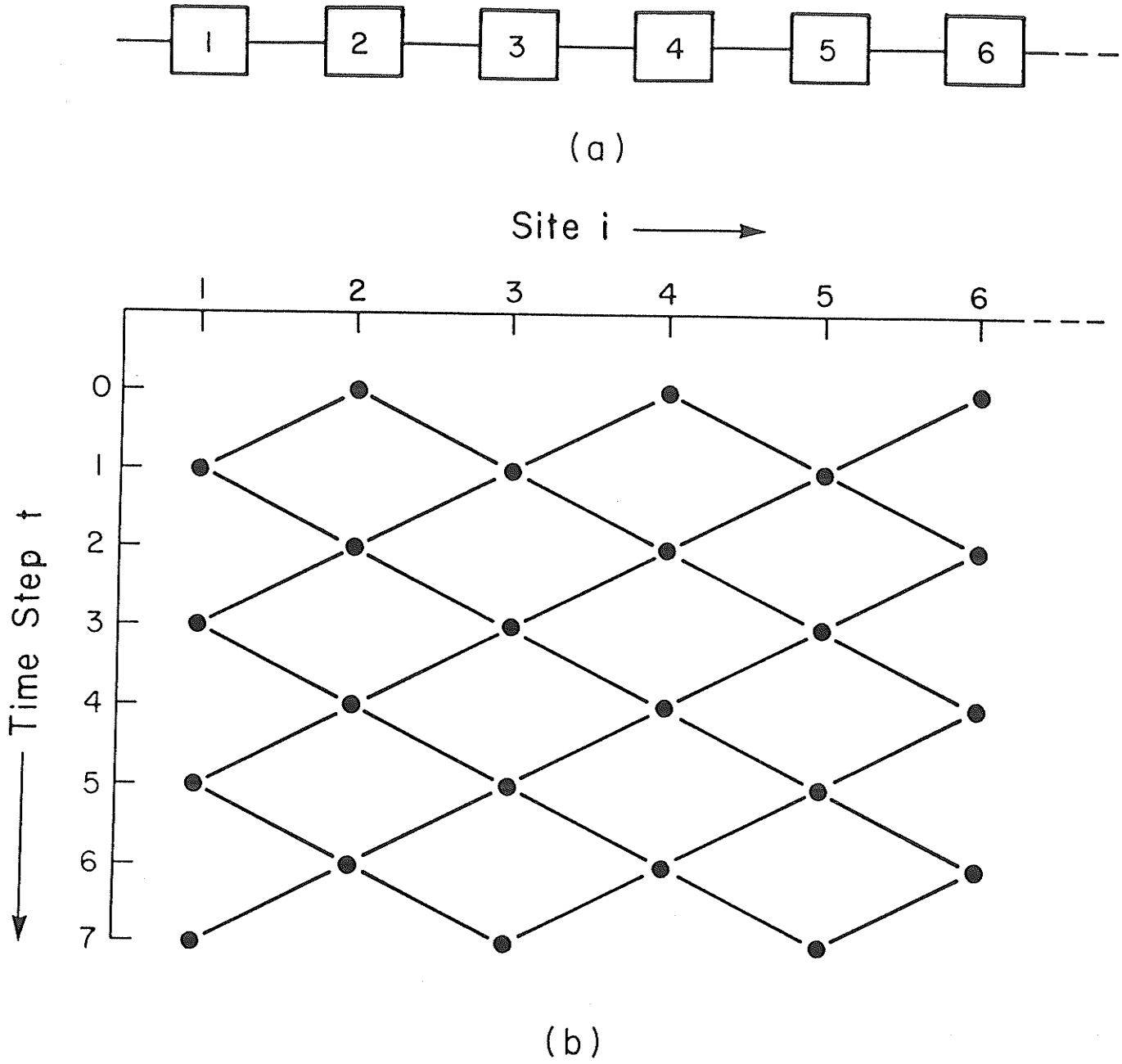


Figure 3.45 : *(a) A one-dimensional cellular automaton. (b) A two-dimensional triangular Ising lattice.*

random behaviour.

3.3.5.1. The Mapping

The exact mapping between one-dimensional stochastic cellular automata and two-dimensional (triangular) Ising lattices has been demonstrated by Domany and Kinzel [Domany1984]. Consider a one-dimensional cellular automaton with site values 0 and 1 and communication restricted to immediate neighbours (Fig 3.45(a)). The boundary conditions may be either periodic (wrap around) or fixed values (0 or 1) depending upon the problem to be simulated. On odd time steps, odd-numbered sites are updated according to the local rule. On even time steps, even-numbered sites are updated. This is an essential countermeasure against the feedback catastrophe which would result if one attempted to update all spins during the same time step.

One can observe the space-time behaviour of the 1-D cellular automaton as the 2-D lattice of Fig. 3.45(b). This figure corresponds to the 2-D triangular Ising model lattice, in which each spin influences (and is influenced by) exactly six neighbour spins. For odd (even) cycles, the odd (even) sites take on the spin value 1 with probabilities $p_1(0,0)$, $p_1(1,1)$, or $p_1(1,0)$. The numbers in brackets are the neighbour spins from the previous time step (the neighbours in a one-dimensional cellular automaton). Also we have that $p_1(0,1)=p_1(1,0)$ and that $p_0(0,0)=1-p_1(0,0)$, etc.

These probabilities are determined from the contribution $E_i=-J(s_{i-1}s_i+s_{i+1}s_i)$ to the total system energy. For a given neighbour configuration, E_i is determined for $s_i=0$ and for $s_i=1$. The probabilities are proportional to $\exp(-E_i/kT)$; the sum of these two factors normalises p_0+p_1 to unity.

In order to generate a series of configurations for the 2-D Ising lattice according to an importance sampling, the organisation of each site value in the 1-D cellular automaton is as shown in Fig. 4.46. For a given temperature, T, for the equilibrium Ising model, the RAM is loaded with the four values corresponding to $p_1(0,0)$, $p_1(0,1)$, $p_1(1,0)$, and $p_1(1,1)$. The RAM selects the appropriate probability value under address control of s_{i-1} and s_{i+1}. This value is compared with that produced by the PRNG and the result (a 1 or 0) is loaded into s_i. For an L site 1-D processor array, every $2L$ time steps or clock cycles, a complete configuration of an $L\times L$ $2-D$ Ising lattice is generated. These values are read from the odd and even sites on alternate cycles, and may be added for a given column in Fig. 3.45(b) by means of serial adders, in order to compute their contributions to the total system magnetisation, M. To obtain an accurate value for M, several thousand lattice configurations must be averaged [Pawley1984].

This architecture yields a speedup of $L/2$ over sequential processing. Of course if, as discussed previously, $O(L^2)$ processing sites are used a further speedup by a factor of L to 2 time steps per complete lattice calculation is possible using $L^2/2$ processing sites. However, this approach, which employs one processor per odd/even lattice site, will use L times more area than the present approach. Thus, the AT metric in both cases is the same if we ignore the extra communication overhead of the L^2

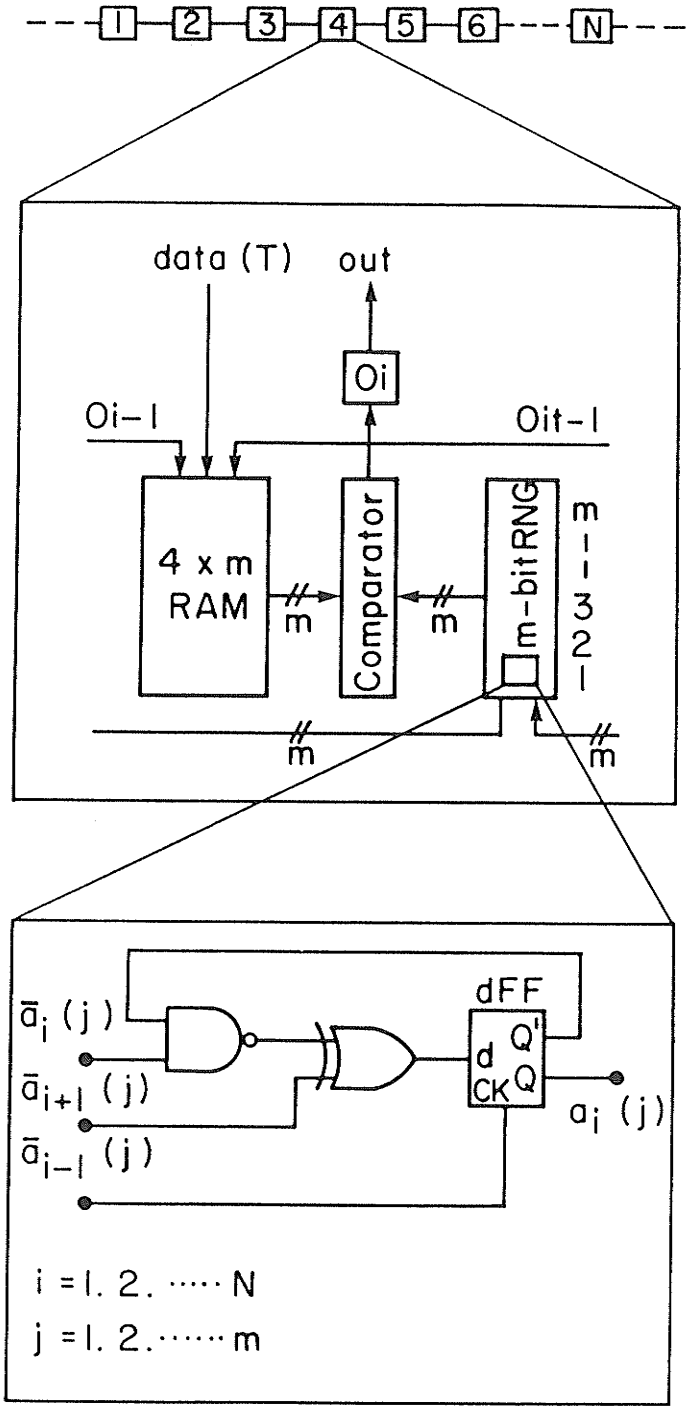


Figure 3.46 : *One processor site of CA-based 1-D Ising computer.*

spin processor implementation. The 1-D processor configuration discussed here results in easily satisfiable wiring requirements. Wilson [Wilson1979] has stated that an Ising lattice problem of interesting size consists of at least 100 × 100 sites, which involves 100 processors using the present approach or 5000 processors using the $L^2/2$ spin processor technique. Therefore, while using $L^2/2$ spin processors leads to

an improved speed performance at the expense of an equivalent increase in area, it has much more complicated wiring requirements and is difficult to make for lattices which are of interesting size. It is more likely that an $O(L^2)$ processor implementation may be appropriate to the importance sampling of the 3-D Ising model, again in the spirit of the Domany-Kinzel mapping.

3.3.5.2. Implementation of the Domany and Kinzel Ising Computer

In addition to the computational advantage of the parallel approach several other points are worth noting. The RAM block in Fig. 3.46 can be shared between two adjacent cells; i.e. the outputs of the RAM can feed into the comparators of both the i and the $i-1$ cells (see Fig. 3.46). This requires two additional address lines from s_i and s_{i-2} . A two-phase clocking scheme will then activate s_{i-1}, s_{i+1} and s_{i-2}, s_i on alternate clock phases. Alternatively, the PRNG can be shared between adjacent sites. There is also provision for applying a temperature gradient across the lattice (left to right), by loading the RAMs across the chip with probability coefficients that change with position in accordance with the desired profile. This will facilitate the analysis of non-equilibrium configurations. One should be aware that the non-equilibrium case of the Ising model is a much more complex problem than the equilibrium case. A further point to note is that if one is not concerned with non-equilibrium configurations then a single RAM for all of the processing site probabilities will suffice, since the probabilities will be the same at each site. The probabilities in this case can simply be routed over the entire chip on m bit busses from the single RAM and each site may select the appropriate probability bus based on its neighbour sites. The bus values are not changed at clock speeds since the equilibrium computation requires unchanging probabilities for each temperature. This approach will lead to a higher density of processing sites since the area used to route the probability busses is considerably less than the combined area of RAMs at each site, especially in double metal CMOS processes.

3.3.5.3. Discussion and Conclusions

The actual layout of this Ising computer on silicon is facilitated by the nearest neighbour communication properties of the CA-based PRNG. Figure 3.47 shows the layout of two PRNG cells in the 3μm double metal CMOS technology described in Chapter 2 using only one layer of metal interconnect. This cell uses 0.138 mm^2 and contains all the necessary connections for the PRNG. In the layout of the Ising computer it was discovered that the layout was easily partitioned into a bit slice architecture with four processing sites per slice. Figure 3.48 shows the layout of a four site Ising processor slice. Note that in this layout the Ising computer permits the use of temperature gradients or non-equilibrium conditions, by providing a RAM at each site. Using a 3 μm single metal process it is possible to have 32 such processing sites on a 4.8 x 4.8 mm. die. This yields only the equivalent of a 32×32 lattice but already provides a speedup of a factor of 16 over conventional sequential processing. By removing the RAM at each site as discussed above and by employing a larger die in a more advanced CMOS technology it is possible to create a layout with over 1000 processors

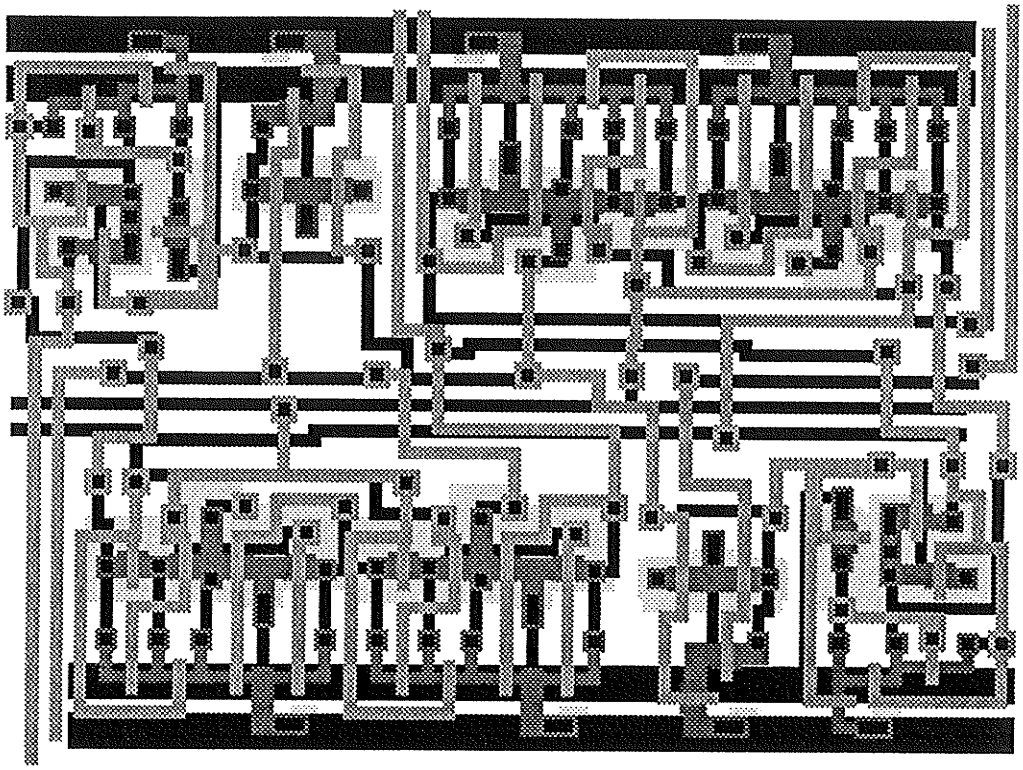


Figure 3.47 : *Layout of two cells in the CA-based PRNG.*

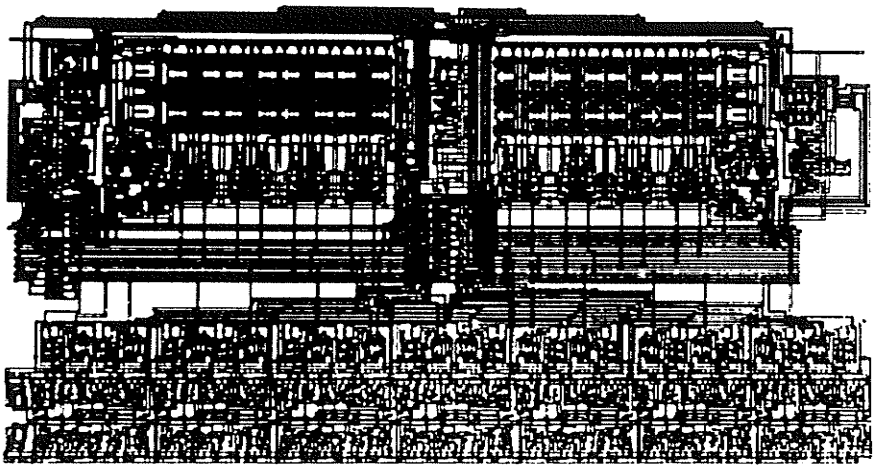


Figure 3.48 : *Layout of a four site Ising processor slice.*

which corresponds to a speedup of at least 3 orders of magnitude over sequential processing. These 1000 processors can be configured as a lattice of 1000×1000 sites or alteratively as ten 100×100 lattices. In the latter case, this would accomplish the sampling of 10 configurations in 200 steps, for an effective rate of 20 steps per

configuration.

Implementation	Update rate [a]	Ref.
Delft Ising System Processor	1.5×10^6	[Hoogland1983]
CDC Cyber 205	22×10^6	[Reddaway1985]
Santa Barbara Ising Model Processor	25×10^6	[Pearson1983a]
ICL DAP	218×10^6	[Reddaway1985]
Manitoba Ising Model Processor I[b]	640×10^6	
Manitoba Ising Model Processor II[c]	20×10^9	
Manitoba Ising Model Processor III[d]	20×10^{12}	

[a] In spin updates per second.
[b] Present 32 site L spin processor configuration.
[c] Projected 1000 site L spin processor configuration.
[d] Projected 1000×1000 site L^2 spin processor configuration.

Table 3.5: *Performance of various Ising model systems.*

Another point to make is that the present scheme may be extended to two-dimensional cellular automata arrays in order to model the 3-D Ising lattice. This affords a computational advantage of $O(L^2)$ as compared with serial computation. In this case there will be additional wiring complexities in the VLSI layout. Finally, at another level of organisation in this Ising computer (beyond that considered here) one expects to be able to employ block-spin renormalisation group methods [Binder1979], [Wilson1979], [Wilson1975], [Niemeyer1974] to process the configurations *in place* in order to recover the system observables. Note that is possible to modify the renormalisation approach considered for the L^2 spin processor architecture discussed previously to operate on this L spin processor architecture.

3.3.6. Conclusions on Ising Computers

The presented Monte Carlo simulations of the Ising model differs from other hardware approaches in that it exploits VLSI to create single chip Ising model processors. These circuits may be used as hardware accelerators for Monte Carlo simulations similar to the use of a floating point accelerator for floating point arithmetic. The speed is mainly achieved by updating the spins at each lattice site on every clock cycle through a novel, and very efficient, parallel random number generation technique. Based on a prototyped 32 site single chip L spin processor both Ising architectures have been designed to operate at a minimum of 20 MHz (in the RAM-less configuration) so that the spins are updated at each processing site every 50 nsec. Therefore, in the final version of these architectures we should be able to update a 1000×1000 spin lattice at a performance of 20×10^{12} spin updates per second for the L^2 spin processor architecture and 20×10^9 spin updates per second for the L spin processor architecture. This substantially improves upon the fastest approach known to

the author of 218 x 10^6 spin updates per second [Reddaway1985]. Table 3.5 compares these architectures with other published implementations.

Chapter 4
Applications to Built-in Self-Test

4.1. INTRODUCTION

Previous chapters have considered the application of cellular automata to problems involving nondeterministic algorithms. Specifically considered were specialised computing architectures for high-speed solutions of ubiquitous computational physics problems such as the Ising model and percolation. Generally these architectures require a large number of uncorrelated pseudorandom numbers to be used at the same time in parallel. In this chapter we consider a very different problem, that of testing VLSI circuits, and specifically of testing using random test vectors. In this problem we are not concerned with generating pseudorandom numbers in parallel but rather with generating pseudorandom numbers at high speed using minimal area. Therefore, while the results of Chapter 2 can still be used we must place a much increased emphasis on the absolute area of silicon used for the PRNG.

4.2. INTRODUCTION TO BUILT-IN SELF-TEST

Design for testability (DFT) techniques attempt to deal with the inherent complexity of the VLSI testing problem by incorporating testability as a primary element of the design process [Williams1983]. A common feature of DFT techniques is the reconfiguration of a sequential circuit so that at test time it can be considered combinational. The sequential circuit latches are used to apply appropriate test vectors and accumulate the resulting response vectors, and are thus themselves also tested indirectly as they verify the combinational logic of the circuit under test. Level Sensitive Scan Design (LSSD) [Eichelberger1977] is an example of such an approach.

In LSSD and similar approaches such as Scan Path [Funatsu1975], Random Access Scan [Ando1980], and Scan/Set [Stewart1977], a test set must still be determined together with the valid responses. At test time each test vector must be serially scanned into the circuit and the corresponding response serially scanned out. While this type of approach greatly reduces the complexity of sequential circuit testing, there are three difficulties:

i) an appropriate test set must be determined, which can require significant computation;

ii) the time required to scan the test vectors in and the circuit responses out can be excessive;

iii) the correct responses must be stored and compared to the observed responses in order to determine if there is a detected fault.

Built-In Self-Test (BIST)[4.1] techniques address these points. In a BIST design the generation and application of the test vectors and the analysis of the resulting response are part of the circuit (or system) under test. As in the scan path techniques a sequential circuit is treated as combinational with the sequential circuit latches used as an integral part of the test. A significant feature of the BIST approach is its low pin overhead which typically consists of two pins; one to put the chip into test mode and one more to deliver the final pass/fail result. In this work the discussion of BIST will be restricted to only those networks which consist of combinational logic and associated sequential latches. In general, BIST refers to any design in which testing is a built-in function of the system. For example, in most modern microprocessors the BIST techniques described here are not applicable since programmable logic arrays (PLAs) and microcoded ROM have replaced the random combinational control logic of earlier designs. However, almost all modern microprocessors provide significant built-in testability features in order to verify correct system operation by using the on-chip processor and memory to run a built-in test program on power up or on user request [Kuban1984]. An interesting paper describing the development of BIST in one company's microprocessor line can be found in [Daniels1985].

A BIST design requires a mechanism for generating an appropriate set of test vectors. For some combinational blocks it is possible to exhaustively apply all the possible input patterns and compare the circuit response to a known *correct* circuit response. An exhaustive input test set can be generated by a simple counter or, alternatively, by a maximal cycle length Linear Feedback Shift Register (LFSR) [Golomb1982].[4.2] For the conventional single stuck-at fault model, first considered in [Eldred1959], an exhaustive test set ensures that every fault will be exercised.

However, if there are more than 20 inputs to the circuit under test, the time to provide the test patterns ($\geq 2^{20}$ per circuit) and the memory to store the circuit responses ($\geq m \cdot 2^{20}$ bits, m = number of circuit outputs) becomes excessive [McCluskey1985a]. For cases where an exhaustive test set is prohibitive a

[4.1] Also referred to as Built-In Test (BIT), self-test, in-situ test, self-verification, or autonomous test.

[4.2] A LFSR with maximal cycle length can produce all input patterns except $00 \cdots 0$. If the all zero pattern needs to be included then a nonlinear feedback shift register which consists of extra logic added to the LFSR can be used.

pseudorandomly selected subset of the possible inputs to the circuit under test is used. This requires an on-chip pseudorandom sequence generator which, in order to reduce the overhead required for BIST, should largely consist of the sequential circuit latches. A technique termed Built In Logic Block Observation (BILBO) [Konemann1979] has emerged as the predominant approach to date and employs a LFSR with maximal cycle length as the pseudorandom sequence generator.

The LFSR-based test pattern generator (L-TPG) is formed by the addition of *exclusive-or* gates to the sequential latches with appropriate control logic so that the latches can perform their normal circuit function as well as be reconfigured for testing. The positioning of the *exclusive-or* gates is given by the primitive polynomial over the Galois Field GF(2) required to form a maximal cycle length LFSR [Golomb1982]. Note that if the length of the L-TPG must be increased (or decreased) due to a design change (i.e. the number of circuit inputs changes) then a completely new primitive polynomial (i.e. LFSR) is required. A further difficulty with L-TPGs is the requirement of a feedback path from the most to the least-significant cell in the LFSR which further complicates the layout of the register.

In this chapter it is shown that a L-TPG has a number of undesirable properties which affect its use in a BIST environment. In particular, it is shown that LFSR generated patterns are not at all appropriate if memory-inducing faults, such as MOS stuck-open faults [Wadsack1978], are being considered, and they provide less than desirable fault coverage for delay or transition faults and other types of AC faults [Barzilai1983].

The new pseudorandom number generators proposed in Chapter 2 are shown to be more appropriate for BIST than conventional LFSR-based generators. In addition to improved randomness properties these new pseudorandom test pattern generators also have implementation advantages in that they require only adjacent-neighbour communication and they are cascadable, i.e. the physical length of the generator[4.3] can be increased or decreased by simply adding or subtracting cells (it should be noted that the area of each cell in a cellular automaton is comparable to a LFSR cell). Therefore, the major redesign required in the case of the LFSR is avoided. This means that a CA-based test pattern generator (C-TPG) is much more appropriate than a L-TPG for incorporation in a computer-aided design (CAD) tool.

BIST also requires a mechanism for reducing the response data to a simple pass/fail result using some form of data compression or compaction. Once again the common suggestion is to employ a LFSR to form a signature of the output data [Frohwerk1977]. The use of a CA-based signature register in place of one based on a LFSR is attractive from a layout perspective. Analysis of the effectiveness of such a CA-based data compactor is reported later in this chapter.

[4.3] For physical length n, where n is the number of cells, or bits, in the test pattern generator, we have a maximum cycle length of $2^n - 1$.

The overall effectiveness of BIST has been the subject of much research [Miller1987] [Williams1986b]. Analysis is usually based upon the assumption that the input test vectors are selected at random. Since BIST must use a pseudorandom number generator to provide the sampling of the input space there will necessarily be discrepancies between the observed and analytical behaviour of a BIST approach. Using the results of Chapter 2 we see that CA-based generators are *more random* than LFSR-based generators, so it is expected that the analytical models of BIST effectiveness are more realistic in a CA-based BIST environment.

4.3. CONVENTIONAL PSEUDORANDOM TEST PATTERN GENERATION

The most popular hardware pseudorandom test pattern generator is the linear feedback shift register. As described in Chapter 2, there are three methods for generating pseudorandom sequences using LFSRs. The method used extensively in the application of LFSRs to BIST is the parallel technique which produces a new test pattern on each clock cycle [McCluskey1985a] [McCluskey1985b]. Recall that the results of Table 2.3 indicate that the parallel LFSR method consistently fails almost all of the random number tests.[4.4]

As indicated in Chapter 2, the most evident failure of the parallel LFSR is in the bit sequence correlation test. The cross-correlation ridge across the entire LFSR of Fig. 2.11(bottom) can be somewhat alleviated by using a feed forward, or multiplying, shift register. In this case the cross-correlation of bits i and j where the tap lies between i and j will be reduced to zero. However, as shown in Fig. 4.1(top), between each tap the cross-correlation is still as in Fig. 2.11(bottom). Note that one cannot simply use a polynomial with $n + 1$ terms to describe the LFSR (i.e. a tap at each bit) since these polynomials are not primitive, and so can yield very short cycle lengths. One could also scramble the output bits of the LFSR so that adjacent bits in the LFSR are not adjacent outputs. However, as shown in Fig. 4.1(bottom), where the bits in the LFSR are randomly mapped to output positions, the cross-correlation ridge of Fig. 2.11(bottom) has now been replaced by cross-correlation spikes throughout the bits of the output pattern. Therefore, scrambling the output bits will not remove the cross-correlation of bits. It should also be noted that it is unlikely that the bits in the LFSR would be randomly mapped to output positions since this would create severe problems in wiring; bits in close proximity in the LFSR would probably still be in close proximity in the output pattern in order to keep the wiring relatively simple.

The cross-correlation of the bit streams in the LFSR yields a number of circuit faults which cannot be detected. For example, in Fig. 4.2 two circuits are given which have faults which cannot be detected by a L-TPG. Figure 4.2(a) shows a simple two input CMOS NAND gate. If we assume a combinational fault model (i.e. faults do not

[4.4] The same tests were made using the standard HP polynomial [HP1978] and similar results were found.

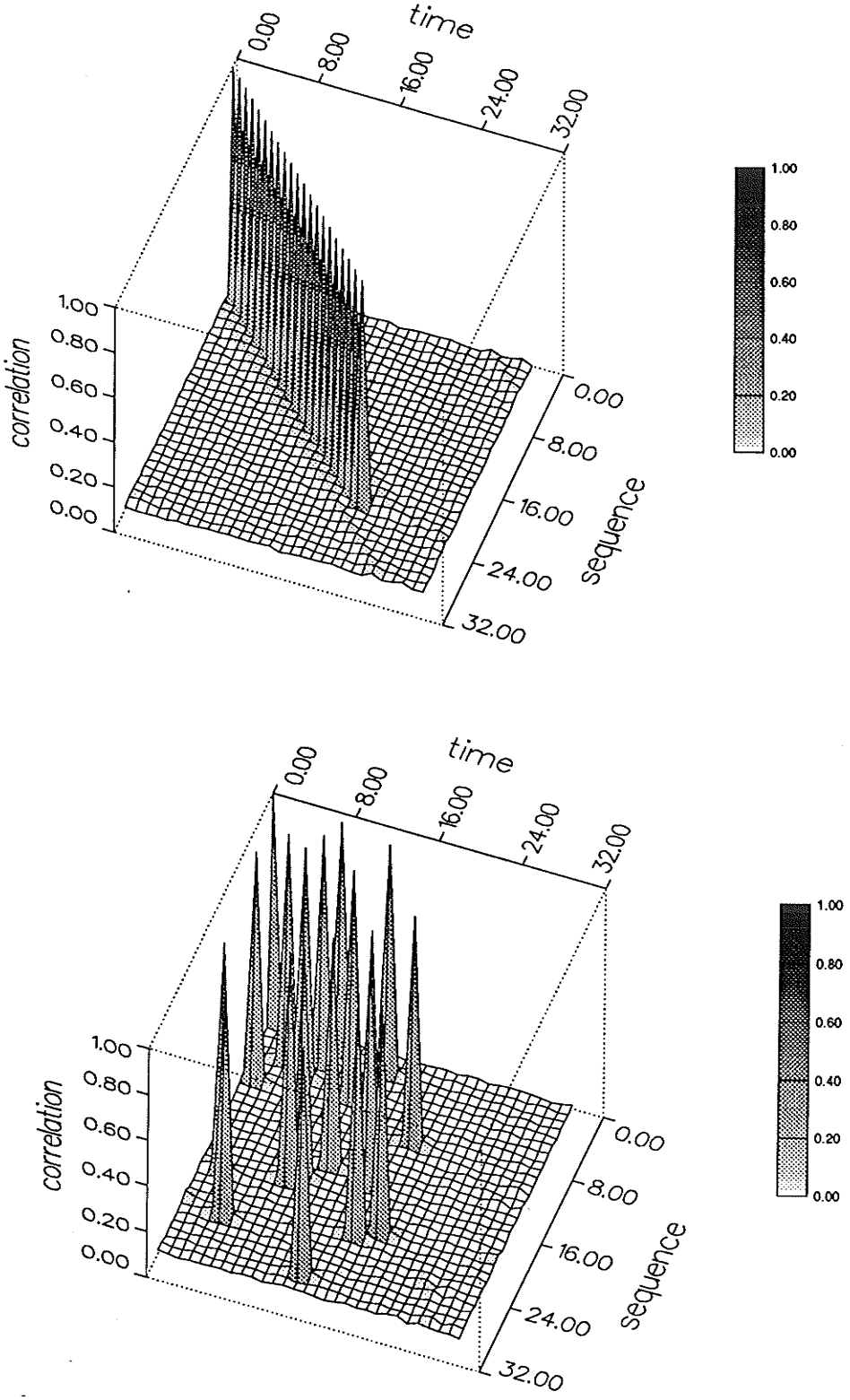


Figure 4.1 : *The cross-correlation of bit sequences in (top) a feed-forward LFSR ($x^{30} + x^{22} + x + 1$) and (bottom) a random mapping of LFSR bits to output sequence bits.*

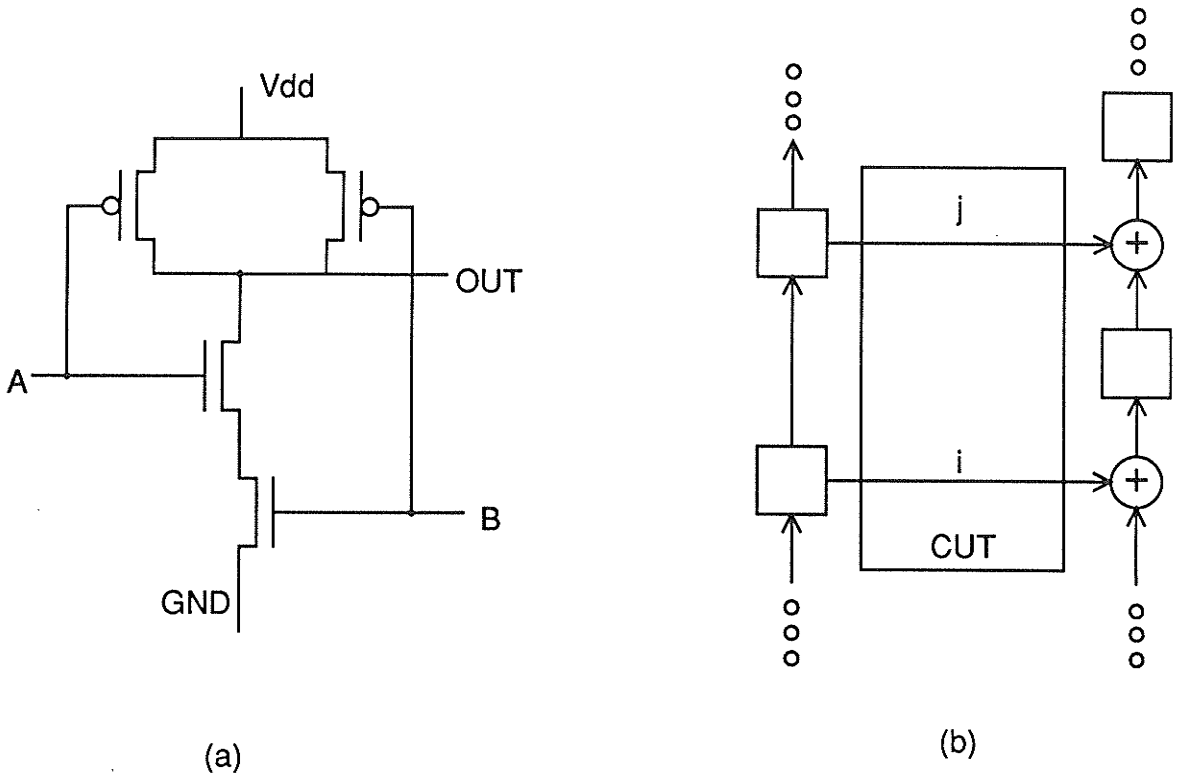


Figure 4.2 : *Two problem circuits for LFSR-based test pattern generators (a) CMOS 2-input NAND gate and (b) a feedthrough network.*

create a sequential circuit from a combinational circuit), such as the stuck-at fault model, then we can completely test the NAND gate using L-TPGs. However, if we consider other circuit faults which do not cause the circuit to act as if a line is stuck-at zero, or one, then the order of test input patterns may be important and the NAND gate may no longer be fully testable. For example [Baschiera1984], an open circuit fault on the B input p transistor induces memory into the circuit since the input A=1, B=0 results in a floating output. This situation, which will hold the last output value until it is eventually brought low by the leakage current, can only be detected by having the input pattern 10 follow input pattern 11. However, this situation can never arise in a L-TPG if the shift direction is from A to B since the value on input A will be on B when the next input pattern is applied. Therefore, one can never completely test a simple two input CMOS NAND gate for stuck-open faults since L-TPGs cannot generate output sequences with an equidistribution of sequence orderings. Note that many other simple circuits can be rendered L-TPG untestable if we use the stuck-open fault model rather than the stuck-at fault model. Furthermore, if we consider CMOS latches and flipflops then the difficulties become even more intolerable. For example, in [Reddy1986] it is shown that stuck-open faults can change static CMOS latches and flipflops into dynamic devices. In addition, it is impossible to detect some stuck-open faults without allowing a sufficient amount of time for the charge to leak from the

output and so reflect the fault (i.e. one cannot test for stuck-open faults at full circuit operating speed). These problems can be overcome by adding some extra transistors to the CMOS latch or flipflop.

In [Mucha1986] it is shown that if a single spacing site is used between each bit in the LFSR (i.e. an N input circuit will require an $N + N$ bit LFSR) then all single stuck-open faults can be detected. However, this is not a desirable situation since the extra N spacing bits will require extra area and also will reduce the operating speed of the L-TPG. As well, the general case of multiple stuck-open faults is still not adequately tested unless we add q spacing sites between each output bit in the L-TPG, where q refers to the number of multiple stuck-open faults to be considered. It should be noted that the number of spacing bits can be drastically reduced if fault simulation is used to identify circuit inputs which are susceptible to cross-correlation in the LFSR. However, rather than be forced to add spacing sites or to do extensive fault simulation to find the minimum required spacing bits, it would be more desirable to have a test pattern generator (TPG) which has an equidistribution of sequence orderings. This is equivalent to requiring that the sequence generated by the TPG pass the *serial* random number tests of Chapter 2.

In Fig. 4.2(b) we consider another very simple failure case [Carter1982]. Here a portion of the circuit under test merely consists of two direct connections from the L-TPG to the LFSR-based multiple input signature analysis register (L-MISR). If we assume that there is no feedback tap in position i of the L-MISR and that the shift direction in the L-TPG is from i to j, then a fault where lines i and j are both stuck-at 0 cannot be detected. This is because of the correlation of L-TPG outputs i and j. The L-MISR will detect the fault when a 1 is applied to input i but, since input j must have the previous value of i, the fault will be cancelled out when the next test pattern is applied. These two examples, along with many others, show a deficiency of L-TPGs and L-MISRs for BIST due to the correlation between adjacent bits. It is recognised that it is quite easy to find a L-TPG which could test these two simple circuits (e.g. scrambling the bits or using spacing bits in the LFSR) but it is just as easy to find another simple CMOS circuit which cannot be tested by the new L-TPG. In addition, to use LFSR-based BIST schemes which adequately test the above problem circuits, one must have prior knowledge of the circuit under test; this is a decided deficiency for a pseudorandom TPG.

These problems lead to the conclusion that conventional L-TPGs, and to a lesser extent L-MISRs, have some major disadvantages and alternative TPGs and MISRs which avoid these problems but utilise comparable area and time would be very desirable devices.

4.4. COMPARISON BETWEEN CA AND LFSR

The approach taken in this work towards hardware pseudorandom number generation for parallel computing architectures can also be applied to BIST. The proposed new BIST structure will be referred to as Cellular Automata-Based Logic Block

Observation (CALBO). Four immediate benefits for the CALBO approach are apparent:

i) the communication is local, being restricted to nearest neighbour cells, which provides freedom from the communication constraints of a LFSR;

ii) the cells are regular and topologically equivalent to one another, in contrast to the increasing complexity of a LFSR layout as the number of sites increases;

iii) routing for the test circuit is no more complicated than the original interconnection of latches (i.e. the topological complexity is contained);

iv) the ability to pass random number tests arises naturally from class 3 (autoplectic) behaviour of cellular automata.

Based upon the autocorrelation functions (cross-correlation at $i = j$) the LFSR, CA rule 30, CA rule 45, and hybrid cellular automata are all observed to display excellent frequency distributions (white spectra). Advantages of cellular automata arise from the reduced cross-correlation associated with cellular automata as compared to the LFSR. As discussed previously, single bit outputs from the LFSR and the above cellular automata are pseudorandom but in most BIST applications the test patterns are generated by considering many bits of the register in parallel. This leads to nonpseudorandom sequences for the LFSR because of the cross-correlation. In addition, there are a number of other problems with sequences generated by considering bits of a LFSR in parallel. Fig. 4.3 illustrates the time evolution of: (top) a rule 30 cellular automaton (length 89, random initial state, cyclic boundary conditions); (middle) a rule 90 and 150 hybrid (length 90, random initial state, null boundary conditions); and (bottom) a LFSR (length 89, random initial state). This figure clearly indicates the much improved cross-correlation properties of C-TPGs over L-TPGs.

The rule 90 and 150 hybrid with single site spacing may be used to provide test patterns for two circuits at the same time by providing alternate site outputs to each circuit under test. This avoids wasting the extra area required for the site spacing and still provides completely uncorrelated test patterns for both circuits. An interesting point to note in this case is that the rule 90 and 150 hybrid may be used as a more complete exhaustive test pattern generator than the LFSR. Bate and Miller [Bate1987] have shown that to exhaustively test an n input CMOS combinational circuit one must apply $n2^{n+1}$ test patterns (the extra factors result from the consideration of single stuck-open faults). In such a situation a $2n$ stage LFSR, or some variation thereof, is usually used. However, this may not be acceptable since there is still considerable cross-correlation between semiadjacent sites and it is possible to miss some faults. On the other hand the unit spaced rule 90 and 150 hybrid has no cross-correlation and so may be able to generate a more complete exhaustive test. That is, by using a rule 90 and 150 hybrid with a site spacing of 1, it is possible to get a percentage of stuck-

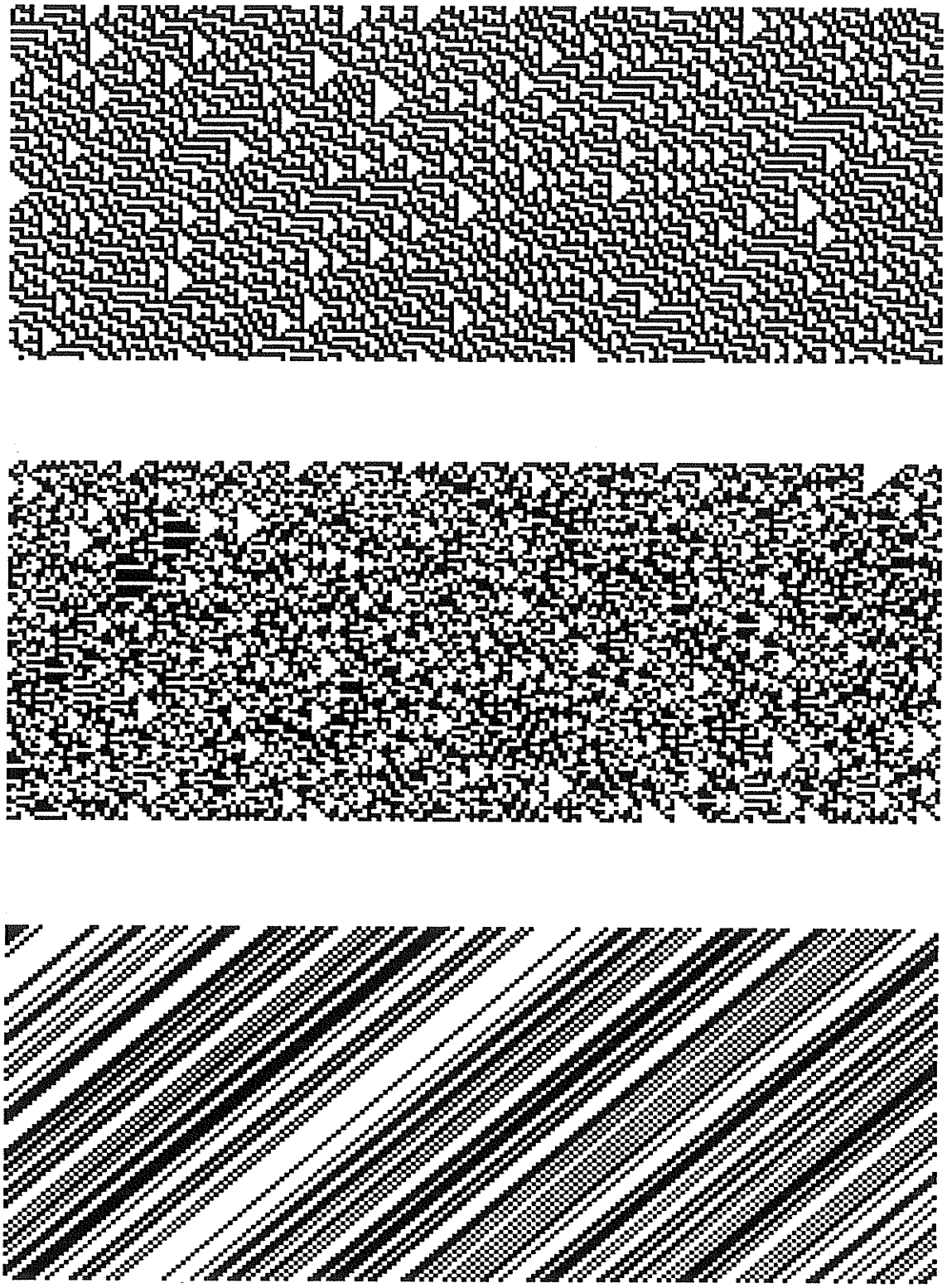


Figure 4.3 : *State - time diagram for (top) CA rule 30 (length 89, random initial state, cyclic boundary conditions); (middle) rule 90 and 150 hybrid (length 90, random initial state, null boundary conditions); and (bottom) LFSR (length 89, random initial state).*

open fault coverage that would not be possible using a $2n$-bit LFSR, provided one is prepared to let the test structure run sufficiently long. In this case, a more complete exhaustive test is best considered pseudoexhaustive and sufficiently long refers to a time roughly equal to, but not excessively longer than, the *burn in* time.

Another advantage of CA-based generators lies in the consideration of the serial random number test results. Here we see that pairs, triples, quadruples, etc... which are crucial to the detection of memory inducing stuck-open faults, are well distributed (i.e. all n-tuples are possible) using any of the CA-based generators, except the rule 90 and 150 hybrid with no site spacing. On the other hand, in the LFSR pairs, triples, quadruples, etc... are not at all well distributed because of the inherent cross-correlation.

4.5. FAULT COVERAGE

Fault coverage is defined here to be the fraction of all possible faults in a circuit under test which are stimulated by a given set of input test patterns. The input test patterns may be generated algorithmically [Roth1967], or may be found by a random, or pseudorandom, sampling of all possible input patterns. Here we will consider input patterns generated by a pseudorandom TPG and will use fault coverage as a measure of the quality of the test set.

In order to derive estimates of fault coverage a fault detectability analysis of the circuit under test must first be made. The detectability of a fault is the number of input patterns which will exercise the fault [Malaiya1984]. Using these results a detectability profile, H, is constructed;

$$H = [h_1, h_2, \cdots, h_N] \tag{4.1}$$

where

N = the number of input patterns, i.e. 2^n for an n input circuit.

h_k = the number of faults with detectability k.

A property of the detectability profile is that

$$\sum_{k=1}^{N} h_k = M \tag{4.2}$$

where

M = the number of possible faults.

Note that in this analysis the stuck-at fault model is usually used since we are considering only faults which can be detected by single patterns.[4.5] The detectability profile

[4.5] Other faults may require a setup pattern followed by the error detecting pattern as in the NAND gate of Fig. 4.2.

can either be found exactly using a technique such as the D-algorithm [Roth1967] or can be estimated by probabilistic analysis such as Savir's cutting algorithm technique [Savir1984]. In any case, it is generally regarded that finding the detectability profile of a circuit is not a trivial task.

The only way to find the exact fault coverage of an input test set, including a pseudorandom set, is through extensive fault simulation. However, this is usually not practical for pseudorandomly generated patterns since the large number of patterns makes simulation prohibitively expensive. Rather, probabilistic arguments are used to derive the *expected fault coverage* $E[C_L]$, where $E[C_L]$ is the expected number of faults that can be detected by a test set of length L divided by the total number of possible faults, M [Wagner1987]. Most analyses of expected fault coverage for pseudorandom TPG use a random sampling model where one samples with replacement from a set of N possible different vectors. For random testing, [Malaiya1984] has shown that the expected fault coverage is

$$E[C_L] = 1 - \sum_{k=1}^{N} (1 - \frac{k}{N})^L \cdot \frac{h_k}{M} \quad . \tag{4.3}$$

In [Wagner1987] a sampling model is used where one samples without replacement from the set of N possible different input patterns. It can be shown, using this model, that the expected fault coverage is

$$E[C_L] = 1 - \sum_{k=1}^{N-L} \frac{\binom{N-L}{k}}{\binom{N}{k}} \cdot \frac{h_k}{M} \tag{4.4}$$

$$\approx 1 - \sum_{k=1}^{N-L} (1 - \frac{L}{N})^k \cdot \frac{h_k}{M} \quad . \tag{4.5}$$

The analysis of [Wagner1987] is more appropriate and accurate for both L-TPG and C-TPG since both TPGs generate each test pattern only once per cycle. However, the random number test results presented in Chapter 2 illustrate the inadequacy of sequences from LFSRs for use as pseudorandom sequences. The analysis of fault coverage given above assumes a pseudorandom test pattern source, so when a LFSR is used as the source it should not be expected that the fault coverage and other calculated measures will be entirely accurate. This does not imply that the fault coverage of the LFSR will be degraded, only that the analysis is not entirely accurate.[4.6] The analysis of [Wagner1987] assumes that each test vector has an equiprobable chance of being selected, yet after one vector has been selected there is not a 1 in $N - 1$ chance of selecting a given vector. Instead, since the LFSR shifts to the right, we are restricting our selection to one vector out of two rather than $N - 1$. This compares to

[4.6] It should be noted that the predictions made using the sample without replacement model correspond closely to actual results derived from computer simulations [Chin1987].

cellular automata where all bit positions are not simply shifts of other bits but functions of other bits, thereby making the behaviour much more apparently unpredictable and hence more pseudorandom. Therefore, the analysis of [Wagner1987] would seem to be more appropriate for better pseudorandom sequence generators such as the C-TPGs described here.

Other measures of test quality such as *test confidence*,

$$c_L = 1 - \frac{\binom{N-L}{k}}{\binom{N}{k}} \quad , \tag{4.6}$$

the probability that a particular fault with detectability k will be detected in a test of length L, *expected test length*,

$$E[L_i] = \frac{N+1}{k+1} \quad , \tag{4.7}$$

for a particular fault of detectability k, and *average test length* to detect all faults if all faults are equally likely,

$$E[L] = \frac{N+1}{M} \sum_{k=1}^{N} \frac{h_k}{k} \quad , \tag{4.8}$$

have also been derived in [Wagner1987] using the sampling without replacement model. It is expected that these measures also hold for C-TPGs.

What is of most concern to a manufacturer of integrated circuits is the probability of shipping a faulty chip. This is usually termed the defect level, DL, and can be shown to be modelled by [Williams1985]

$$DL = 1 - Y^{(1 - E[C_L])} \tag{4.9}$$

where

Y = process yield, i.e. the probability of manufacturing a good chip.

The effect of process yield on the required fault coverage in order to have a given defect level is illustrated in Fig. 4.4. Notice the sensitivity of defect level, and thereby the required fault coverage, to the process yield. Typically the fault coverage versus the number of random test patterns results in a curve such as that shown in Fig. 4.5 [Williams1985]. Therefore, given a desired defect level and the process yield, one can determine the necessary test pattern length, using either Fig. 4.5 or Eqns 4.5 and 4.8. This calculation is normally employed in designs using L-TPG but also will hold if the design uses a C-TPG; we maintain that this analysis is more accurate for C-TPGs than L-TPGs since a truly pseudorandom TPG is assumed.

It was previously noted that the above analysis assumes a stuck-at fault model. If we use a more complicated fault model incorporating stuck-open or ac faults then, as we have seen in the example problem circuits of Fig. 4.2, the fault coverage of the L-TPG is lower than the C-TPG. This is also shown in [Barzilai1983] where an empirical

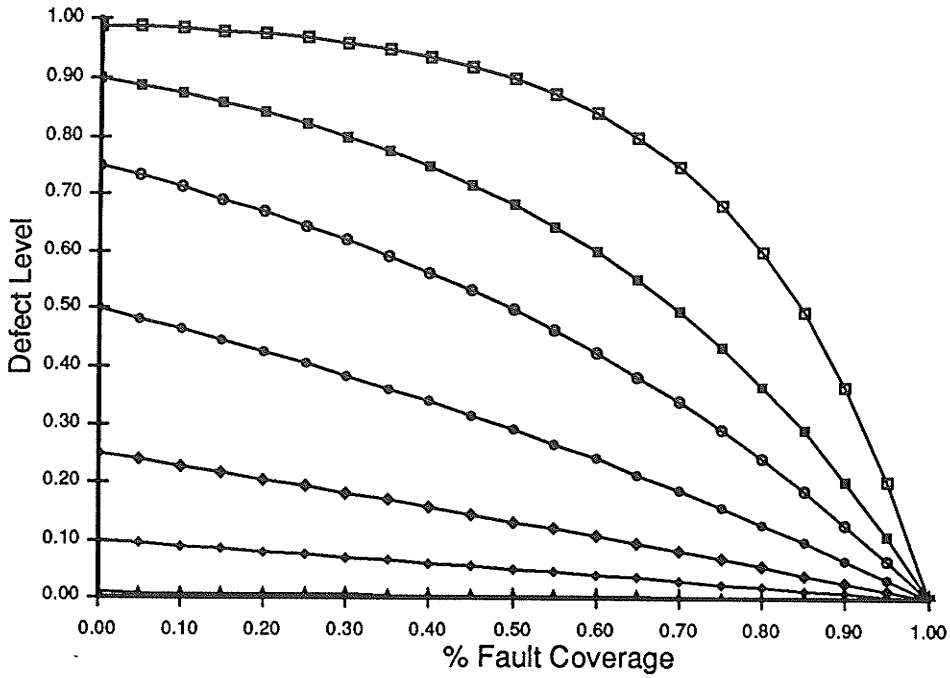


Figure 4.4 : *Defect level as a function of fault coverage and process yield. Curves are for* $Y = 0.01$ *(unfilled squares),* $Y = 0.10$ *(filled squares),* $Y = 0.25$ *(unfilled circles),* $Y = 0.50$ *(filled circles),* $Y = 0.75$ *(unfilled diamonds),* $Y = 0.90$ *(filled diamonds), and* $Y = 0.99$ *(unfilled triangles).*

analysis of a 27 input, 7 output circuit with 262 ac faults showed that a L-TPG could not achieve 100% fault coverage for slow-to-rise and slow-to-fall faults. However, 100% fault coverage could be achieved by a truly pseudorandom TPG. In Chapter 2 it was shown that the cellular automata under consideration for use here as C-TPGs, produce the equidistributed pairs necessary for the detection of these faults. Therefore, C-TPGs should provide improved fault coverage as compared to L-TPGs for ac faults.

Finally, we note that the probability of detecting ac faults is much lower than that of traditional stuck-at fault detection. For example, in [Waicukauski1987] an empirical analysis of the *Brglez-Fujiwara circuits* [Brglez1985] reveals that many more random test patterns are required for ac faults than for dc faults in order to reach the point at which one cannot detect new faults, i.e. only undetectable faults remain.

4.6. WEIGHTED PATTERN GENERATION

An interesting possibility for CA-based generators is that it may be possible to generate pseudorandom sequences which have a statistical weighting to one region of the test pattern space. This area has been investigated for L-TPGs by several authors.

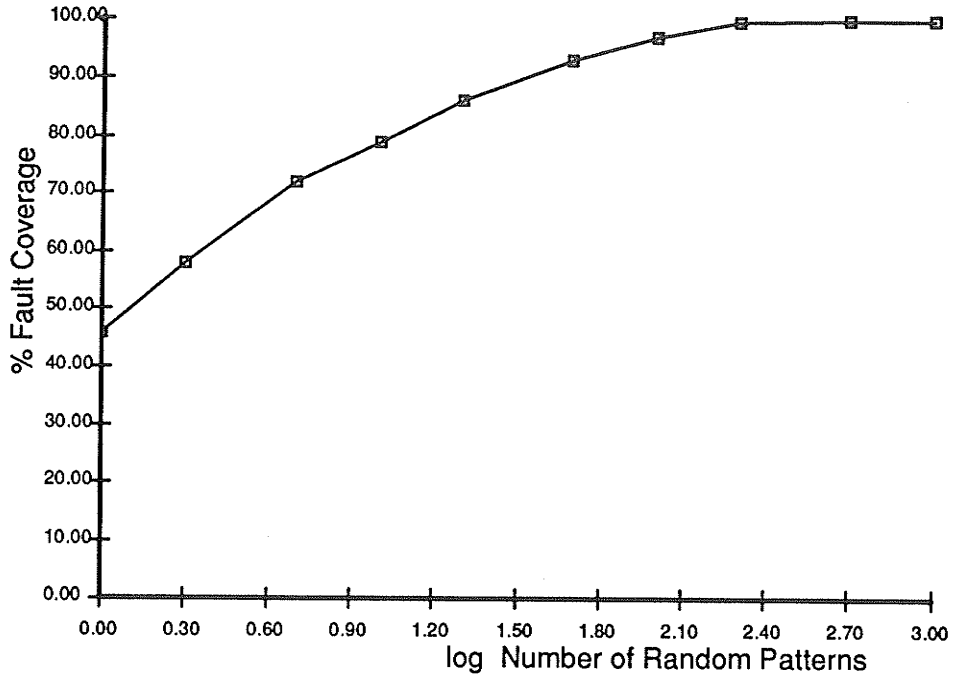


Figure 4.5 : *Typical fault coverage as a function of random test pattern length.*

However, either the resulting BIST structure is very large [Schnurmann1975] or the selection of weight probabilities is very limited [Chin1984]. We know that the rule 90 and 150 hybrid exhibits weighted pattern generation properties for longer sequence lengths. However, it would be enlightening to check if other cellular automata exhibit this property and whether or not the weighting probability can be easily adjusted.

In Appendix C a set of tables are given showing the weights of 1 bits emanating from various positions in all primative one-dimensional cellular automata. It can be seen that the weight probabilities vary depending on the CA rule used and in some cases on the position in the cellular automaton. However, we must also ensure that the generated patterns also conform to a weighted pseudorandom sequence. This means that the random number tests of Chapter 2 would need to be modified in order to properly test a sequence which is not equidistributed. It is not the intent of this work to perform an in-depth study of weighted pattern generation using C-TPGs but rather it is important to note that C-TPGs may be capable of this type of test pattern generation for BIST. Furthermore, it is also possible that some form of hybrid or synthesised cellular automaton may provide the desired function. However, in general the generation of a cellular automaton conforming to a given pseudorandom statistical weighting is a very complex problem.

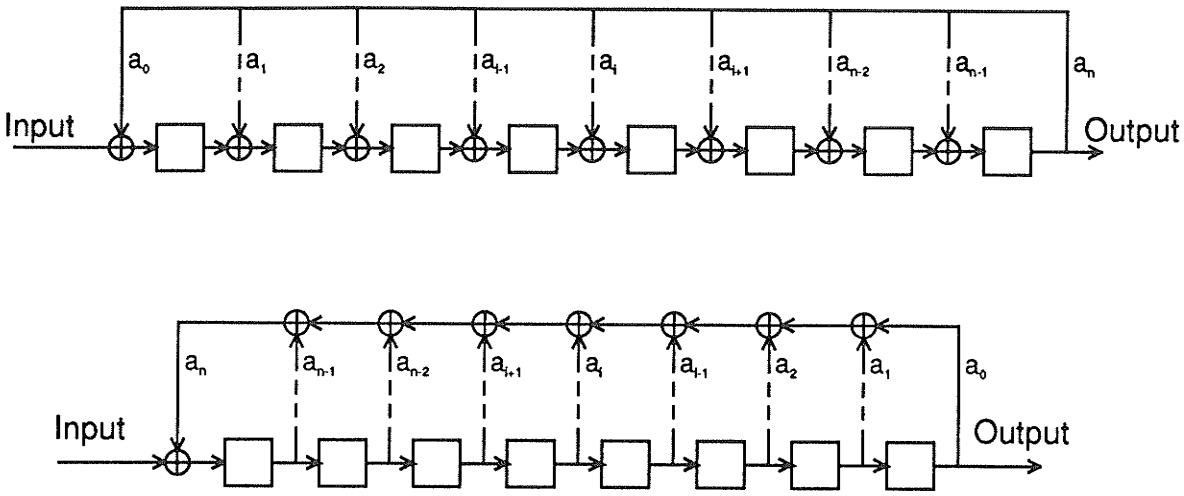


Figure 4.6 : *Two techniques of polynomial division using LFSRs; (top) internal exclusive–or type. (bottom) external exclusive–or type.*

4.7. SIGNATURE ANALYSIS

As mentioned in the introduction, BIST requires some mechanism to reduce the volume of output response data from the circuit under test to a simple pass/fail result. The most popular output test data compaction method uses error detection and correction techniques for cyclic redundancy check (CRC) codes. These error detecting and correcting circuits make extensive use of LFSRs and were developed in the late 1950s and early 1960s [Prange1957] [Meggitt1961]. They are well understood and are thoroughly explained in the algebraic coding theory literature as *syndrome detection* [Lin1983] [Peterson1972] and in the digital testing literature as *signature analysis* [4.7] [Fujiwara1985] [Tsui1987]. In the present work we focus our examination on the use of cellular automata for signature analysis (SA) in BIST. To facilitate the discussion concerning the proposed CA-based signature analyzers a brief summary of conventional LFSR-based signature analyzers follows.

4.7.1. LFSR-Based Signature Analysis

The conventional signature analysis circuit uses a LFSR to implement a repeated polynomial division of a binary input data stream. In Fig. 4.6 the two methods of implementing polynomial division using LFSRs are shown. Here we will consider an m bit LFSR to be implemented using its characteristic polynomial

$$C(x) = c_m x^m + c_{m-1} x^{m-1} + \cdots + c_1 x + c_0 \quad . \tag{4.10}$$

We also consider an n bit binary data stream to be represented by the polynomial

[4.7] This term was introduced by Hewlett-Packard to describe the first commercial product using these principles [Chan1977].

$$P(x) = p_{n-1}x^{n-1} + p_{n-2}x^{n-2} + \cdots + p_1x + p_0 \quad , \tag{4.11}$$

where the high order coefficient, p_{n-1}, enters the LFSR first, followed on successive clock cycles by the lower order coefficients. Finally, we define the quotient of $\frac{P(x)}{C(x)}$ to be

$$Q(x) = \frac{P(x)}{Q(x)} = q_{n-m-1}x^{n-m-1} + q_{n-m-2}x^{n-m-2} + \cdots + q_1x + q_0 \, . \tag{4.12}$$

If the contents of the LFSR is initially set to zero then after n clock cycles the quotient, $Q(x)$, has appeared at the output (most significant coefficient first). For the circuit of Fig. 4.6(top) the contents of the LFSR after n clock cycles corresponds to the remainder or signature, $S(x)$, of the division. Therefore, we have

$$P(x) = Q(x) \cdot C(x) + S(x) \quad . \tag{4.13}$$

For the purposes of this work we will consider $P(x)$ to correspond to the output from a single output fault free network under stimulus from a given set of input patterns. We then define the output from the same circuit in which a fault has occurred to be $P_E(x)$, where we assume that $P(x) \neq P_E(x)$ and that the same set of input patterns for both the fault free and faulty circuit were used. The error polynomial, $E(x)$, will be defined to be the difference between $P(x)$ and $P_E(x)$, so

$$P_E(x) = P(x) + E(x) \quad . \tag{4.14}$$

An undetectable error is one for which the signatures of $P(x)$ and $P_E(x)$ are the same and in such a situation the signature analyzer is said to have produced an *aliased* output. The fact that aliasing can occur is indicative of the fact that SA is a compact testing method [Losq1976], i.e. some error information is lost, as opposed to a data compression technique where no information is lost.

In the case of the LFSR circuit of Fig. 4.6(bottom) the output also corresponds to the quotient of $\frac{P(x)}{C(x)}$ but the final contents of the LFSR is **not** the remainder of the division. However, the final contents of the LFSR is also called the signature of $P(x)$ because it is isomorphic to the actual remainder and so the two signature analyzers share the important property of Theorem 4.1.

Theorem 4.1: *[Frohwerk1977] If $S(x)$ is the signature of $P(x)$ using the circuit of either Fig. 4.6(top) or Fig. 4.6(bottom) then $S_E(x)$, the signature of $P_E(x)$, will equal $S(x)$ if and only if $E(x)$ is a multiple of $C(x)$, the characteristic polynomial of the LFSR.*

Proof: [Smith1980] For the signature analyzer of Fig. 4.6(top) we see from Eqn. 4.13 that

$$P(x) = Q(x) \cdot C(x) + S(x) \quad . \tag{4.15}$$

From Eqns. 4.14 and 4.15 we have

$$P_E(x) = P(x) + E(x) = Q_E(x) \cdot C(x) + S_E(x) \quad . \tag{4.16}$$

If $S(x) = S_E(x)$ and we substitute for $P(x)$ then

$$Q(x) \cdot C(x) + S(x) + E(x) = Q_E(x) \cdot C(x) + S(x) \quad . \tag{4.17}$$

so

$$E(x) = (Q(x) + Q_E(x)) \cdot C(x) \quad . \tag{4.18}$$

Thus, $E(x)$ is of the form $A(x) \cdot C(x)$, i.e. a multiple of $C(x)$.

For the signature analyzer of Fig. 4.6(bottom) a more difficult analysis is required and the reader is referred to [Meggitt1961].

Using Theorem 4.1 a measure of the effectiveness of signature analysis for the detection of single bit errors (i.e. $E(x)$ has just one nonzero term) can be derived.

Theorem 4.2: *[Frohwerk1977] A signature analyzer using a LFSR based on a characteristic polynomial with two or more nonzero terms will detect all single bit errors.*

Proof: [Smith1980] If $C(x)$ has two or more nonzero terms then any multiple of $C(x)$ must also have two or more nonzero terms. In Theorem 4.1 we showed that $E(x)$ must be a multiple of $C(x)$ in order for $S(x)$ to equal $S_E(x)$. Therefore, if $E(x)$ has only one nonzero term it cannot be a multiple of $C(x)$, and must therefore be detectable.

Several other measures of the effectiveness of LFSR signatures have been proposed in the literature and will be stated without proof.

Theorem 4.3: *[Frohwerk1977] [Smith1980]. For a data stream of length n, if all possible error patterns are equally likely, the probability that a length m signature analyzer will not detect the error is*

$$\frac{2^{n-m} - 1}{2^n - 1} \quad . \tag{4.19}$$

Note that as $n \rightarrow \infty$, the probability of missing an error becomes 2^{-m}. However, the theorem is not as strong as one would like. For example, unlike Theorem 4.2, the choice of characteristic polynomial used in the LFSR has no bearing on its error-

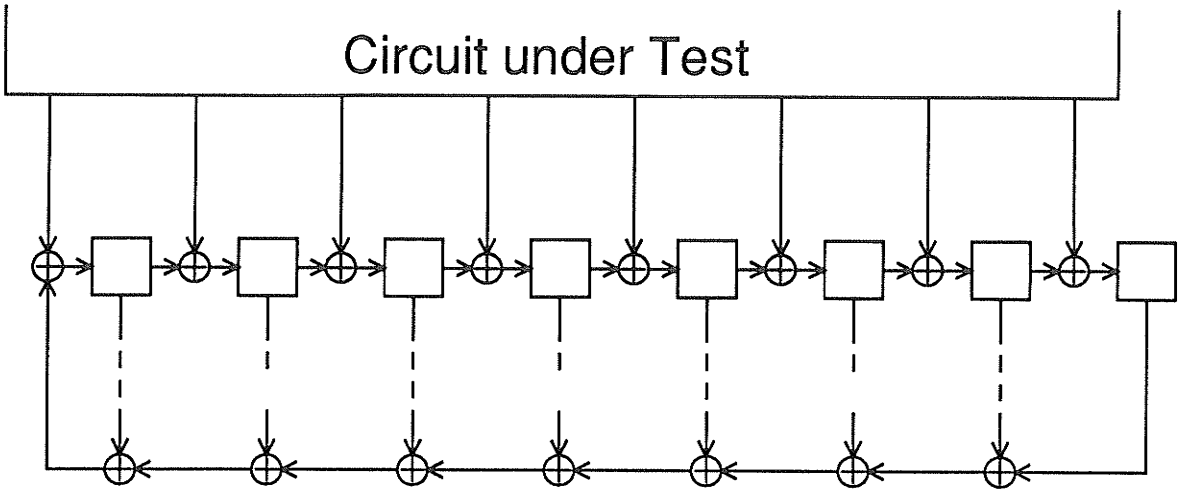


Figure 4.7 : *Multiple input signature analysis register*

detecting capability for multibit errors. This means that a LFSR with a simple $C(x)$ such as x^m is as effective as a LFSR with a much more complicated $C(x)$ such as $x^{16} + x^9 + x^7 + x^4 + 1$ [HP1978]. However, intuitively one would think that a more complicated LFSR would lead to better signature analysis. This discrepancy between Theorem 4.3 and intuition lies in the fact that, unlike communication channels where transmission errors can be assumed independent, output errors in digital systems due to circuit faults are not independent [Smith1980]. Notice that Theorem 4.3 indirectly assumes error independence since all errors are equally likely. However, deriving measures of error detection when errors are dependent is difficult. [Smith1980] has examined specific types of dependent errors such as burst errors and errors due to repeated use, but further analysis of other types of dependent errors is not generally available in the literature. Therefore, in the light of the difficulty in performing a general analysis of dependent errors it would appear that fault simulation of the circuit under consideration is the only means of truly determining the effectiveness.

Up to this point only signature analyzers operating on a single stream of data have been considered. Most practical circuits have many outputs. To form a signature of the output from a multiple output circuit one could: 1) place a separate signature analyzer on each circuit output; 2) direct each output in turn to the LFSR using a multiplexer and form the signature on the resulting single bit data stream [Benowitz1975]; or 3) one could use a multiple input signature analyzer register (MISR) such as that shown in Fig. 4.7 [Benowitz1975]. Option 1 requires excessive area since a register will be required at each output, while option 2 suffers a time penalty in converting the parallel output data to a serial data stream. Presently the MISR circuit of Fig. 4.7 is considered to be the most efficient means of producing a signature of a multiple bit data stream. Several analytical measures of error detecting capability for MISR circuits have been proposed.

Theorem 4.4: *[Bhavsar1981] Consider an r output circuit and assume that all possible error sequences are equally likely. If one forms a signature on N output vectors from the circuit using an m-bit LFSR then the probability of failing to detect an error is*

$$\frac{2^{rN-m}-1}{2^{rN}-1} \quad . \tag{4.20}$$

Proof: We see that for N r-bit output vectors there are $2^{rN}-1$ possible error sequences. It can be shown that an m-bit LFSR which implements a primitive characteristic polynomial maps all possible input sequences equally over the 2^m possible signatures. Therefore, the number of error sequences which cause aliasing is $\frac{2^{rN}}{2^m-1}$ since there are 2^m-1 possible signatures which do not cause aliasing. Thus, the probability of failing to detect an error sequence is

$$\frac{2^{rN-m}-1}{2^{rN}-1} \quad . \tag{4.21}$$

Notice that once again all errors are assumed to be equally likely and that as $N \rightarrow \infty$ the probability of aliasing becomes 2^{-m}. The analysis of [Carter1982] requires almost no assumptions of the output error patterns.

Theorem 4.5: *[Carter1982] If N input test patterns are applied randomly then the probability of aliasing in an m-bit LFSR is less than $4/N$ where we assume that $m > \log_2(N-1)$.*

This result makes no assumptions on the input test pattern generator other than that the test set be applied in random order. Therefore, the results of Theorem 4.5 are equally valid for both a specially selected set of test vectors (i.e. through fault simulation) or a randomly-generated set of test vectors. However, many consider this upper bound to be overly pessimistic. Carter himself has indicated his personal belief that the probability is $\leq N^{-1}$ but has been unable to show this. A further point to note is that once again no restrictions are placed on the characteristic polynomial of the LFSR. An empirical study on 41 typical circuits [Muzio1987] shows that the probability of aliasing is greatly dependent on the charteristic polynomial. As well, for complicated polynomials such as the 16-bit HP polynomial, the results of [Muzio1987] seem to agree with an analytical study [Williams1986a] that the actual probability of aliasing is much closer to 2^{-m} rather than $4/N$.

4.7.2. CA-Based Signature Analysis

We proceed to describe and propose some measures on the effectiveness of signature analysis using cellular automata.

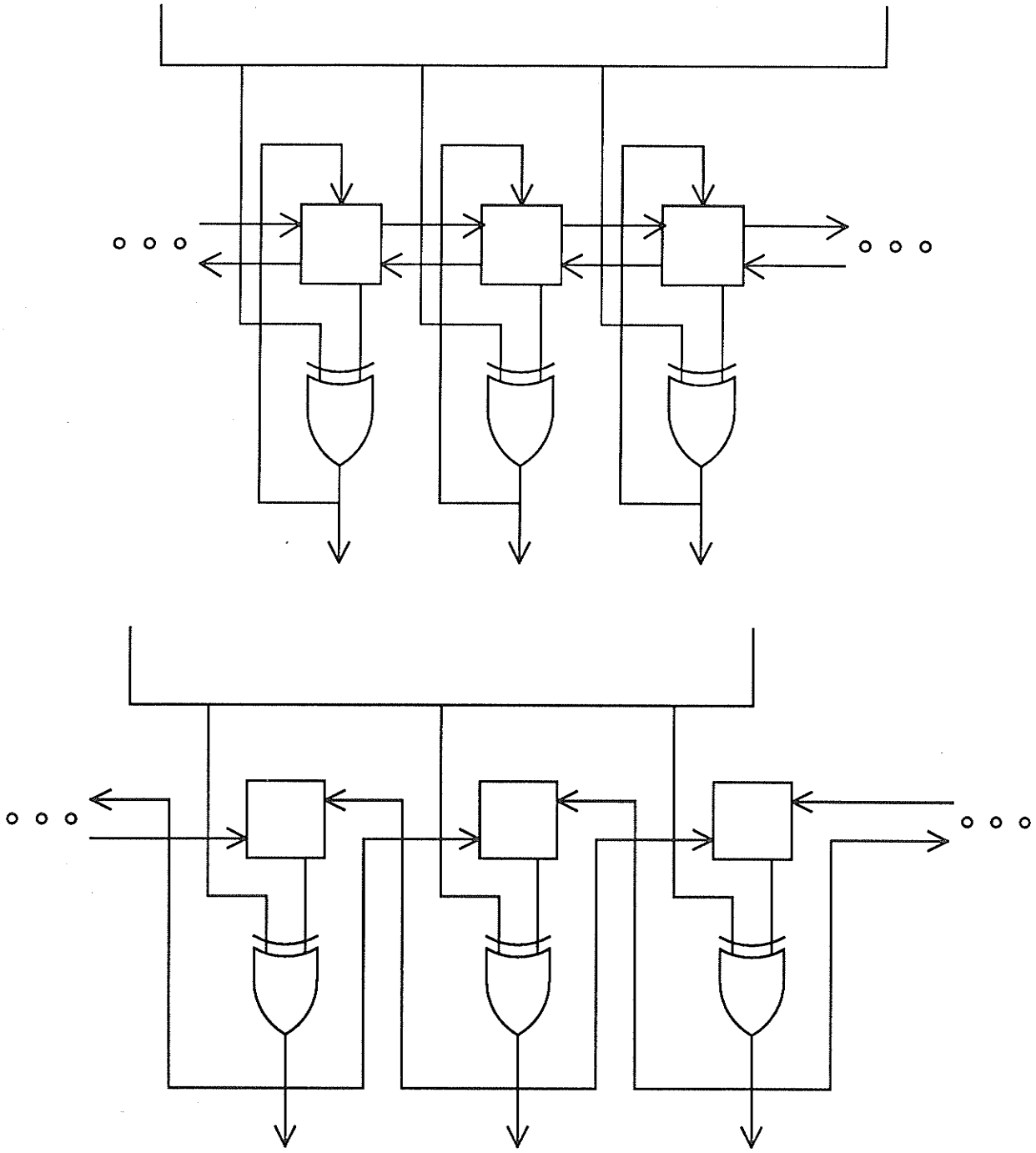


Figure 4.8 : *Two techniques of SA using CA-based MISRs;*
(top) method 1: $R(t+1) = R(t)' \oplus O(t+1)$.
(bottom) method 2: $R(t+1) = (R(t) \oplus O(t))'$.

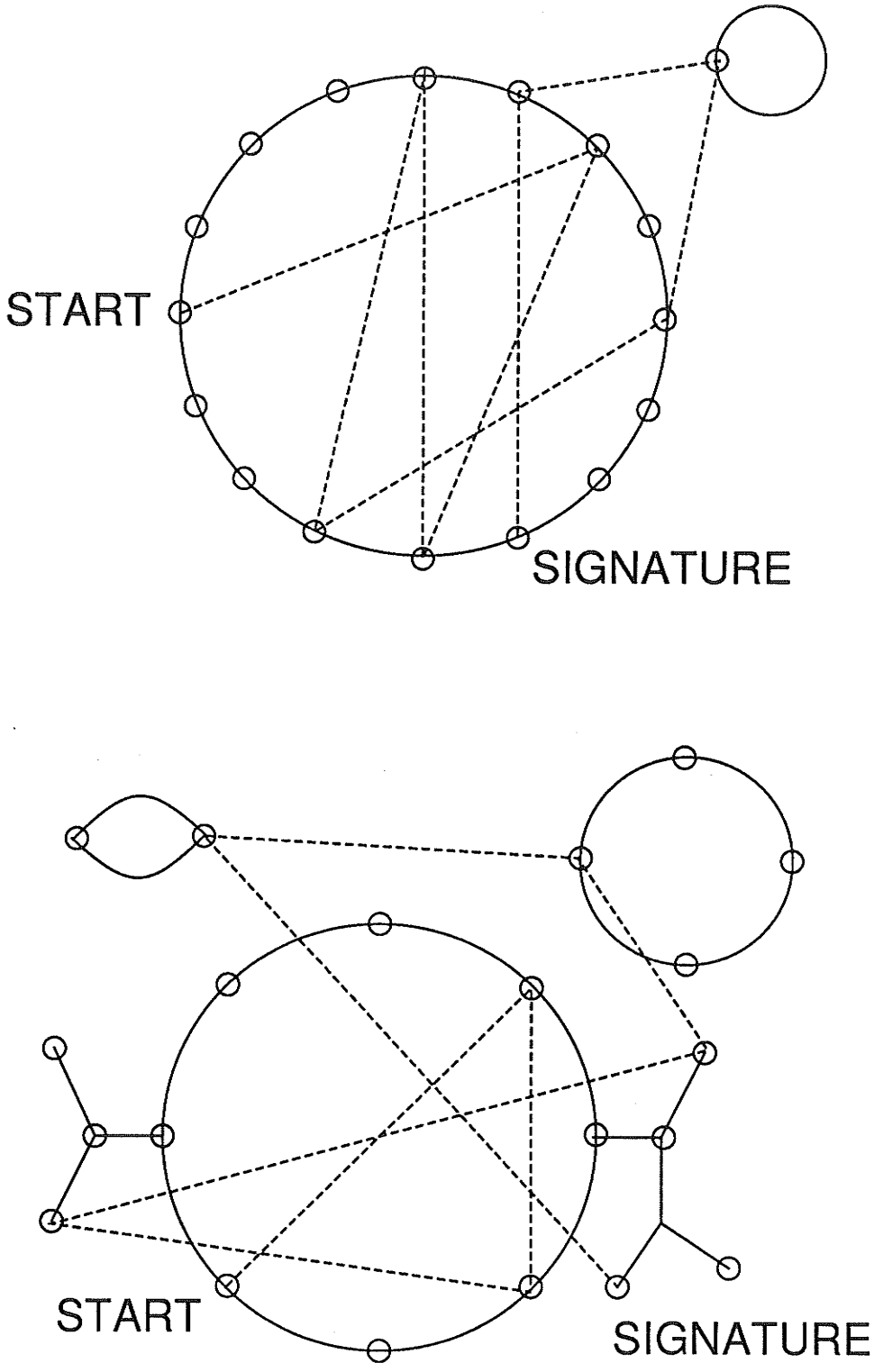


Figure 4.9 : *Comparison of the rule 90 and 150 hybrid, CA rule 30, and the LFSR for signature analysis. Global state transition diagrams and data compression permutations for (top) rule 90 and 150 hybrid and LFSR, and (bottom) CA rule 30.*

4.7.2.1. Two Methods of C-MISR Implementation

The nearest neighbour communication properties required for implementing elementary one-dimensional cellular automata allow the consideration of several different techniques of SA. Here we will only consider four methods but it is acknowledged that other techniques are possible and may, after due consideration, prove to be more satisfactory. The first two techniques are shown in Fig. 4.8. In Fig. 4.8(top) we see that the signature is formed by updating the cellular automaton and then *exclusive-or*ing the current state at each site in the cellular automaton with the corresponding output from the circuit under test. This means that the number of sites required in the cellular automaton is equal to the number of outputs from the circuit under test. Notice that this is directly analogous to conventional L-MISRs. The second technique, shown in Fig. 4.8(bottom), is similar except that here we first *exclusive-or* each site with the corresponding circuit output and then increment the cellular automaton. These two methods can be described algebraically by the following equations.

Method 1:

$$R(t+1) = R(t)' \oplus O(t+1) \quad . \tag{4.22}$$

Method 2:

$$R(t+1) = (R(t) \oplus O(t))' \quad . \tag{4.23}$$

where

$R(t)$ = cellular automaton contents at time t.
$O(t)$ = circuit output at time t.
$R(t)'$ = the incremented value of the cellular automaton contents at time t.

In the BILBO circuit, signature analysis is usually performed via a multiple input signature analysis technique in a similar manner to the generation of cyclic redundancy codes. It can easily be shown that the CALBO circuit is also capable of compacting data as a consequence of the excellent pseudorandom number generation capabilities of cellular automata. The properties of the multiple input LFSR, the rule 90 and 150 hybrid, and CA rule 30 or 45 of importance in signature analysis using methods 1 and 2 are summarised in Fig. 4.9. Figure 4.9(top) represents the state transition diagram associated with a LFSR having a maximal cycle length. Data compaction and subsequent signature generation is accomplished by mixing output vectors for the system under test with the present LFSR state. This effectively permutes the state transition diagram as indicated by the dashed lines. As the multiple inputs to the L-MISR are nondeterministic, each point in the state space is equiprobable including the null, or zero, state. A similar mapping is appropriate for the maximal length rule 90 and 150 hybrid discussed previously.

The situation for CA rule 30 is somewhat different in that it is not a maximal cycle length state machine, but rather one whose state transitions consist of trees and subcycles as indicated in Fig. 4.9(bottom). We suggest that for signature analysis, this state diagram may also be appropriate because of the nondeterministic nature of the

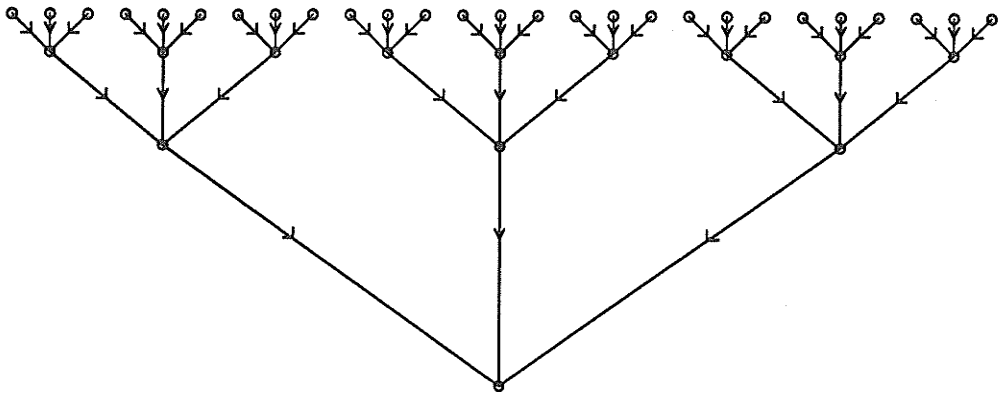


Figure 4.10 : *A directed m-ary tree. Here* $m = 3$.

outputs from the system under test. Transitions readily occur between states in separate cycles (semigroups), so that all states in the state space of CA rule 30 are also equiprobable. Hence data compaction capabilities should be similar for cellular automata and LFSRs. Note that in the case of the second LFSR problem circuit of Fig. 4.2(b), the missed error occurred because the signature analyzer cancelled out the error when the LFSR shifted. If a CA-based MISR (C-MISR) had been used the error would not be cancelled since each bit in the C-MISR is a function of the incoming information and its three neighbour bits. This contrasts with the L-MISR where each bit is a function of the incoming information and only one neighbouring bit. However, different cellular automata will yield different aliasing probabilities and thereby indicate which cellular automata are more suited for use in C-MISRs.

Unlike the L-MISR there is a very large number of different cellular automata-based structures which can be used as C-MISRs. This variety allows for a great deal of flexibility in C-MISR design, but at the same time it is impossible to do an exhaustive examination of all the possibilities and identify the best overall implementation. One is also severely constrained by the lack of formal algebraic techniques for examining the evolution of many CA implementations. Here some results will be presented which attempt to identify properties that should be possessed by candidate cellular automata for SA, using methods 1 and 2.

We first consider the evolution of states for different cellular automata. The behaviour of many cellular automata is similar to that shown in Fig. 2.26 in that there are many cycles and paths to the cycles. This compares to the most common LFSR-based MISRs which have one large cycle of $2^n - 1$ states and the zero state as a one-cycle. The major difference for SA is that in cellular automata which have an evolution of states similar to that of CA rule 30 we have individual states with more than one predecessor. The ramifications of this can be most readily determined by studying SA on a directed m-ary tree. Here we define a directed m-ary tree to be a tree in which each state other than the so called *garden of eden* states has m possible predecessors directed towards itself, as shown in Fig. 4.10, and the mapping of the N

possible states is randomly ordered.

Lemma 4.1: *Consider three binary words A, B, and X, then $A \oplus X \neq B \oplus X$, if $A \neq B$.*

Proof: Consider binary values, a, b, and x, where $a \neq b$, then $a \oplus x \neq b \oplus x$ by nature of the bitwise *exclusive-or* operation. Therefore, if A, B, and X are binary words, where $A \neq B$, then $A \oplus X \neq B \oplus X$ since the *exclusive-or* operator operates only on single bits.

Lemma 4.2: $1-x \leq e^{-x}$.

Proof: For all x, e^{-x} can be represented by the alternating power series $1 - x + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots$. We see that alternating terms decrease in magnitude so that $1-x \leq e^{-x}$.

Theorem 4.6: *Using signature methods 1 and 2 on a directed m-ary tree the probability that the signatures of two random sequences, which differ in only one element, are different T increments after the differing element, is*

$$\leq e^{\frac{-T(m-1)}{N}} . \tag{4.24}$$

Proof: Let $SA_1(t)$ and $SA_2(t)$ be the signatures after t elements of sequences 1 and 2, respectively. Let the differing element occur in element s, then $SA_1(s-1) = SA_2(s-1)$ but $SA_1(s) \neq SA_2(s)$. Here we are considering an m-ary tree where each branch has the form of Fig. 4.11. The only way in which SA_1 and SA_2 can become equal under method 1 at step t is for the *exclusive-oring* at step $t-1$ to have permuted both $SA_1(t-2)'$ and $SA_2(t-2)'$, to the same set of m possible successors states. The probability that this occurs is $\frac{m-1}{N}$. Note that they cannot be permuted to the same predecessor state by Lemma 4.1. Therefore, the probability that SA_1 and SA_2 remain different at each step is $1 - \frac{m-1}{N}$. Under method 2 for SA_1 and SA_2 to become equal on step t, the *exclusive-oring* at step t must permute both $SA_1(t-1)$ and $SA_2(t-1)$ to the same set of m predecessor states. The probability that this occurs is the same as for method 1. This occurs independently for each step under methods 1 or 2 so, T steps after the differing element, the probability that SA_1 and SA_2 have, at some step, been permuted to the same set of predecessors using methods 1 or 2 is

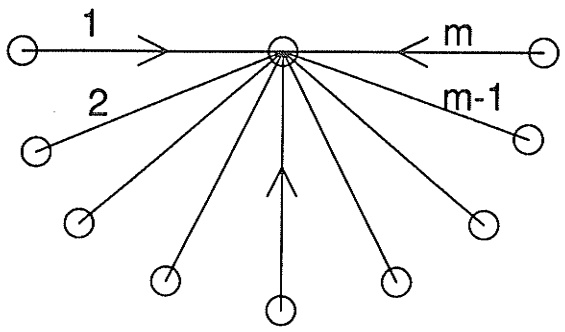


Figure 4.11 : *A degree m branch in a directed m-ary tree.*

$$(1 - \frac{m-1}{N})^T \quad . \tag{4.25}$$

Therefore, by Lemma 4.2 and setting $x = \frac{m-1}{N}$ we can say that, using methods 1 and 2 on a directed m-ary tree the probability that the signatures of two sequences differing in only one element remain different, T elements after the differing element, is

$$\leq e^{\frac{-T(m-1)}{N}} \quad . \tag{4.26}$$

Of course, most CA rules do not implement directed m-ary trees or even directed binary trees for that matter but using these techniques we can form more exact measures for differing cellular automata. We first consider a general tree structure such as that shown in Fig. 4.12. Here we have branches with varying degrees including those with degree zero (i.e. states with no successor) which are sometimes referred to as *graveyard states*. We can tabulate the number of branches of degree i as N_i. For example, Fig. 4.12 has two branches of degree 2 so $N_2 = 2$. Notice that the total number of states, N, equals

$$N_0 + \sum_{N_i > 0} i \cdot N_i. \tag{4.27}$$

Theorem 4.7: *Consider a general tree structure with N_i branches of degree i, then the probability that the signatures using methods 1 and 2 of two random sequences differing in only one element are still different, T steps after the differing element, is*

$$\left\{1 - \sum_{N_i>0} \frac{i(i-1)N_i}{N^2}\right\}^T \quad . \tag{4.28}$$

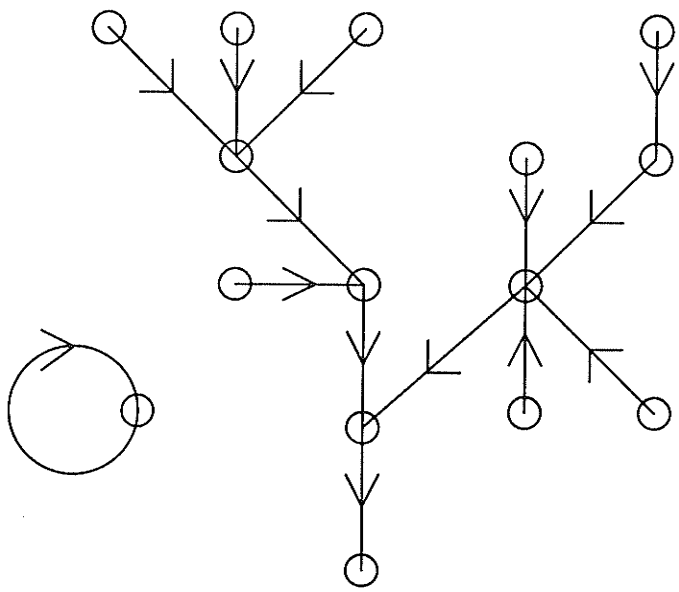


Figure 4.12 : *A general directed tree structure. Here* $N_0 = 1$, $N_1 = 2$, $N_2 = 2$, $N_3 = 1$, *and* $N_4 = 1$.

Proof: As in Theorem 4.6, let SA_1 and SA_2 be the signatures of sequences 1 and 2, respectively and let the differing element occur in element s. Since each sequence is random, the values must be equidistributed. It can be shown that the *exclusive-or* operator is linear and so, SA_1 and SA_2 are permuted equally around the tree by the circuit outputs. Therefore, using method 1, the probability that $SA_1(t)'$ has been permuted to a branch of degree i is $\frac{i \cdot N_i}{N}$, for $i \geq 1$ and $\frac{N_0}{N}$ for $i = 0$. For SA_1 and SA_2 to become equal at some step $t > s$ using method 1 both $SA_1(t-2)'$ and $SA_2(t-2)'$ must be permuted to the same branch. Given that SA_1 has been permuted to a branch of degree i the probability that SA_2 will be permuted to the same branch is $\frac{i-1}{N}$. Therefore, the probability that $SA_1(t-2)'$ and $SA_2(t-2)'$ are permuted to the same branch equals the sum over all the branches of degree ≥ 1 of the probability that $SA_1(t-2)'$ is permuted to a branch of degree i multiplied by the probability that $SA_2(t-2)'$ is permuted to the same branch, i.e.

$$\sum_{\text{all branches}} \frac{i \cdot N_i}{N} \cdot \frac{i-1}{N} = \sum_{N_i > 0} \frac{i \cdot N_i}{N} \cdot \frac{i-1}{N} \tag{4.29}$$

$$= \sum_{N_i > 0} \frac{i(i-1)N_i}{N^2} \quad . \tag{4.30}$$

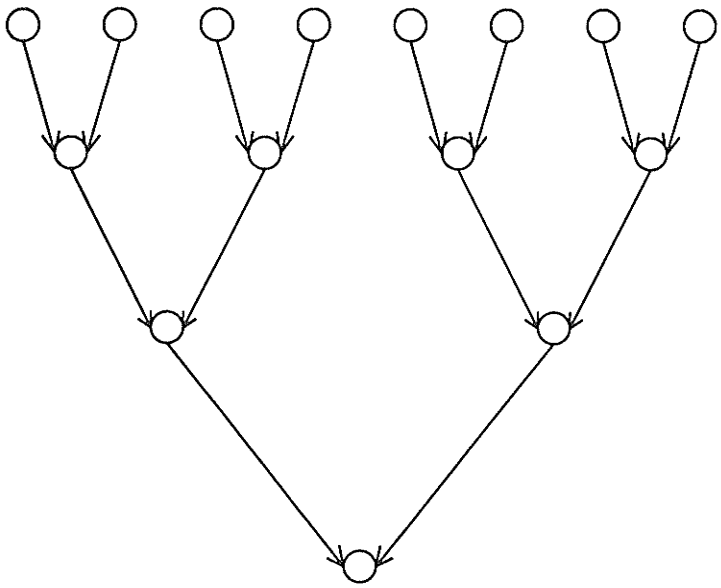


Figure 4.13 : *A directed binary tree with 7 branches*

Therefore, the probability that $SA_1(t-2)'$ and $SA_2(t-2)'$ are not permuted to the same branch (i.e. remain different) is

$$1 - \sum_{N_i > 0} \frac{i(i-1)N_i}{N^2} \quad . \tag{4.31}$$

A similar argument can be constructed for SA method 2.

The probability of SA_1 and SA_2 remaining different at each SA step is independent, so T steps after the differing element the probability that SA_1 and SA_2 are still different is

$$\left\{1 - \sum_{N_i > 0} \frac{i(i-1)N_i}{N^2}\right\}^T \quad . \tag{4.32}$$

A quick check to verify that Theorems 4.6 and 4.7 agree shows that if we use the directed binary tree of Fig. 4.13 then the probability of not aliasing on each step according to Theorem 4.7 is $1 - \frac{2 \cdot 1 \cdot 7}{14^2} = \frac{13}{14}$ which equals the result of $1 - \frac{1}{14} = \frac{13}{14}$ from Theorem 4.6.

Using Theorem 4.7 we can now find the probability of aliasing on single bit errors for any CA-based MISR provided we know the state transition diagram. For example, a 4 bit cyclic rule 30 cellular automaton has the state transition diagram of Fig. 4.14(top), yielding a probability of not aliasing of $(1 - \frac{8}{256})^T = 0.969^T$, while a 4 bit null

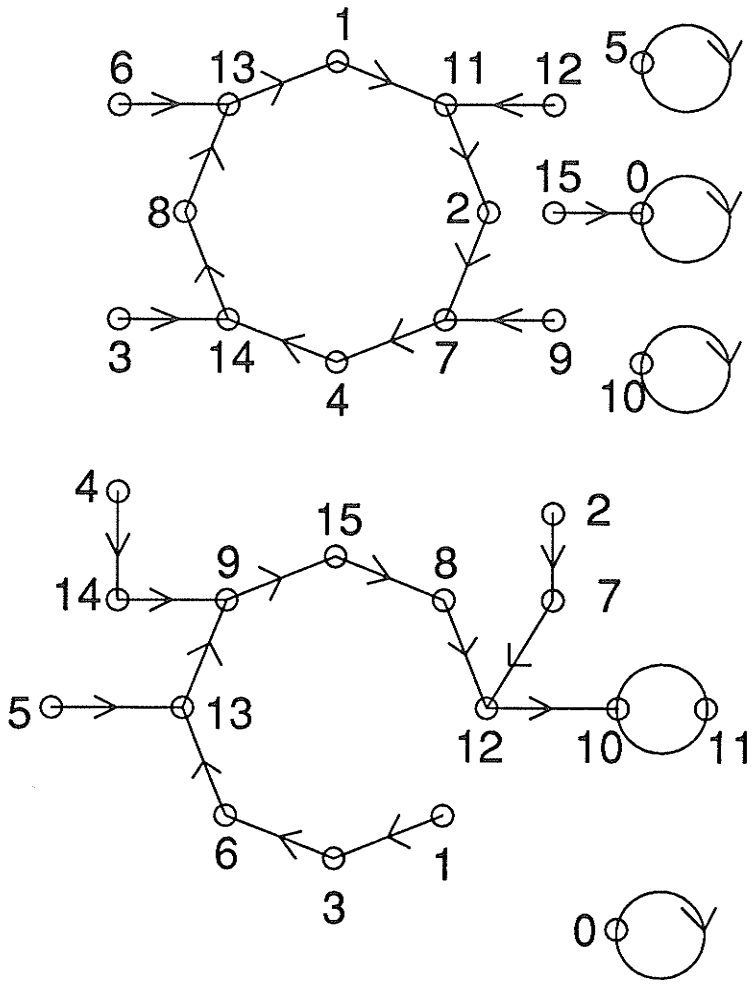


Figure 4.14 : *State transition diagram for (top) 4 bit rule 30 cellular automaton with cyclic boundary conditions. (bottom) 4 bit rule 30 cellular automaton with null boundary conditions.*

boundary rule 30 cellular automaton with state transition diagram, as shown in Fig. 4.14(bottom), has a probability of not aliasing of $(1 - \frac{6}{256})^T = 0.977^T$. If we examine more CA-based MISRs we see that a trend establishes itself. This trend indicates that cellular automata with the fewest degree ≥ 2 branches in the state transition diagram have the smallest aliasing probability. In fact, we see from Theorem 4.7 that if the state transition diagram consists only of unary branches (i.e. cycles) then the aliasing probability for single bit errors is zero. A number of CA rules lead to such behaviour and are listed in Table 4.1.

Rule	Equation	Boundary	Length
204	$a_i(t)$	all	all
51	$\overline{a_i(t)}$	all	all
60	$a_{i+1}(t) \oplus a_i(t)$	null	all
195	$\overline{a_{i+1}(t) \oplus a_i(t)}$	null	all
102	$a_i(t) \oplus a_{i-1}(t)$	null	all
153	$\overline{a_i(t) \oplus a_{i-1}(t)}$	null	all
90	$a_{i+1}(t) \oplus a_{i-1}(t)$	null	4,6,8, $\cdots$
165	$\overline{a_{i+1}(t) \oplus a_{i-1}(t)}$	null	4,6,8, $\cdots$
150	$a_{i+1}(t) \oplus a_i(t) \oplus a_{i-1}(t)$	null	4,6,8, $\cdots$
105	$\overline{a_{i+1}(t) \oplus a_i(t) \oplus a_{i-1(t)}}$	null	4,6,8, $\cdots$
240	$a_{i+1}(t)$	cyclic	all
15	$\overline{a_{i+1}(t)}$	cyclic	all
170	$a_{i-1}(t)$	cyclic	all
85	$\overline{a_{i-1}(t)}$	cyclic	all
150	$a_{i+1}(t) \oplus a_i(t) \oplus a_{i-1}(t)$	cyclic	all
105	$\overline{a_{i+1}(t) \oplus a_i(t) \oplus a_{i-1(t)}}$	cyclic	all
101	$(a_{i+1}(t) \cup a_i(t)) \oplus a_{i-1(t)}$	cyclic	5,7,9, $\cdots$
154	$(a_{i+1}(t) \cup a_i(t)) \oplus \overline{a_{i-1(t)}}$	cyclic	5,7,9, $\cdots$
89	$(a_{i+1} \cup \overline{a_i(t)}) \oplus a_{i-1}(t)$	cyclic	5,7,9, $\cdots$
166	$(a_{i+1} \cup \overline{a_i(t)}) \oplus \overline{a_{i-1}(t)}$	cyclic	5,7,9, $\cdots$
75	$a_{i+1}(t) \oplus (\overline{a_i(t)} \cup a_{i-1}(t))$	cyclic	5,7,9, $\cdots$
180	$\overline{a_{i+1}(t) \oplus (\overline{a_i(t)} \cup a_{i-1}(t))}$	cyclic	5,7,9, $\cdots$
45	$a_{i+1}(t) \oplus (a_i(t) \cup \overline{a_{i-1}(t)})$	cyclic	5,7,9, $\cdots$
210	$\overline{a_{i+1}(t) \oplus (a_i(t) \cup \overline{a_{i-1}(t)})}$	cyclic	5,7,9, $\cdots$
90, 150 hybrid	$a_{i+1}(t) \oplus a_{i-1}(t)$ $a_{i+1}(t) \oplus a_i(t) \oplus a_{i-1}(t)$	null	4,6,10,12, $\cdots$

Table 4.1: *CA rules implementing a cyclic group for both null and cyclic boundary conditions.*

Theorem 4.8: *Using SA methods 1 and 2 we will always have a different signature for two sequences differing in only one element provided that the MISR's rule of operation forms a cyclic group.*

Proof: A cyclic group's state transition diagram consists solely of unary branches so, from Theorem 4.7, we see that the probability of aliasing given a single differing element is zero. Therefore, the two sequences must have different signatures.

A second proof for Theorem 4.8 can be made by using induction on Lemma 4.1.

Let the differing element occur in element s then $SA_1(s) \neq SA_2(s)$. Using method 1 on the next signature step we have by Lemma 4.1

$$SA_1(s+1) = SA_1(s)' \oplus O(s+1) \neq SA_2(s)' \oplus O(s+1) = SA_2(s+1) \ . \quad (4.33)$$

Assume that $SA_1(t-1) \neq SA_2(t-1)$, $t = s+k > s+1$, then by Lemma 4.1

$$SA_1(t) = SA_1(t-1) \oplus O(t) \neq SA_2(t-1) \oplus O(t) = SA_2(t) \ . \quad (4.34)$$

Therefore, the statement is true for $t = s$, $s + 1$, and $s + k$ so by induction, it is true for all $t > s$.

For SA method 2 a similar inductive argument to that for SA method 1 can be used. Thus, if the MISR's rule of operation forms a cyclic group, SA using methods 1 and 2 will always yield a different signature for two sequences differing in only one element.

Notice that Theorem 4.8 also holds for conventional L-MISRs.

The general theorem for m-ary trees and the specific example for CA rule 30 show that implementing a MISR using a circuit which contains states with multiple predecessors, such as that which occurs in many CA rules, is poor. This is especially true when we compare the results to Theorem 4.8 where we see that guaranteed detection of single differences, or errors, is possible using LFSRs and certain CA rules. Therefore, only those rules implementing cyclic groups should be used for MISRs since for single errors both C-MISRs and L-MISRs will provide the same fault detection capabilities. Thus, for single errors, it has been shown that using a C-MISR built from a CA rule of Table 4.1 provides equivalent SA properties to those of a L-MISR.

For multiple errors we must consider how the additional errors will affect the probability of not aliasing given in Theorem 4.7.

Theorem 4.9: *If we consider the general directed tree structure of Theorem 4.7 then the probability that the signatures using methods 1 and 2 of two random sequences differing in two randomly placed elements are still different T_2 steps after the second differing element is*

$$\left[\sum_{N_i>0} \frac{i(N-i)N_i}{N^2} - 1\right] A(T_2)\, A(T_1-T_2) + A(T_2) \quad . \tag{4.35}$$

where

$$A(T) = \left\{1 - \sum_{N_i>0} \frac{i(i-1)N_i}{N^2}\right\}^T \quad .$$

T_j = time since the j'th differing element.

Proof: Using Theorem 4.7 we know that on a general directed tree structure, the probability that the signatures, using methods 1 and 2, of two random sequences differing in only one element, are still different T steps after the differing element, is

$$\left\{1 - \sum_{N_i>0} \frac{i(i-1)N_i}{N^2}\right\}^T \quad . \tag{4.36}$$

Consider a second differing element which occurs at some time T after the first differing element. There are three possibilities at this juncture:

i) if the signature is still different then there is a probability that the second differing element permutes the signature to that of the other sequence;

ii) if the signature is still different and the second differing element does not permute the signature to that of the other sequence then the probability of still having a differing signature T_2 steps later is as in Theorem 4.7;

iii) If the two signatures have become the same then the second differing element will act as in Theorem 4.7.

The probability of the signatures still being different T steps after the first differing element is as in Eqn. 4.36. The probability of the second differing element permuting the signature to the other sequence's signature is, summing over all the branches, the probability of the other signature being in a branch of degree i at step T times the probability of permuting the differing signature to a specific branch of degree i, i.e.

$$\sum_{\text{all branches}} \frac{i \cdot N_i}{N} \cdot \frac{i}{N} = \sum_{N_i>0} \frac{i^2 N_i}{N^2} \quad . \tag{4.37}$$

Therefore, the probability for the occurrence of instance i) is

$$\sum_{N_i>0} \frac{i^2 \cdot N_i}{N^2} \cdot \left\{1 - \sum_{N_i>0} \frac{i(i-1)N_i}{N^2}\right\}^T \tag{4.38}$$

where i and N_i are defined as in Theorem 4.7.

Conversely, we see that the probability of not permuting to the other sequence's signature is

$$\sum_{N_i>0} \frac{i \cdot N_i}{N} \cdot \frac{N-i}{N} = \sum_{N_i>0} \frac{i(N-i)N_i}{N^2} \quad . \tag{4.39}$$

Therefore, the probability of instance ii) occurring is

$$\sum_{N_i>0} \frac{i(N-i)N_i}{N^2} \cdot \left\{1 - \sum_{N_i>0} \frac{i(i-1)N_i}{N^2}\right\}^T \quad . \tag{4.40}$$

For instance ii) the probability of the two signatures remaining different after this point in time is as in Theorem 4.7 with $T = T_2$ since the C-MISR is a first order system. For instance iii) we see that the probability of the two sequences having become the same is

$$1 - \left\{1 - \sum_{N_i>0} \frac{i(i-1)N_i}{N^2}\right\}^T \quad . \tag{4.41}$$

As in instance ii), the probability of the two signatures remaining different after this point in time is as in Theorem 4.7 with $T = T_2$.

The total probability of the two signatures remaining different is the sum of instances ii) and iii). Thus, the probability of the signatures using methods 1 and 2 of two random sequences differing in two randomly placed elements still being different T_2 steps after the second differing element is

$$\sum_{N_i>0} \frac{i(N-i)N_i}{N^2} A(T_1-T_2)\, A(T_2) + (1 - A(T_1-T_2))\, A(T_2) \tag{4.42}$$

where $A(t)$ and T_j are defined as above.

Collecting terms we see that we can reduce Eqn. 4.42 to

$$\left[\sum_{N_i>0} \frac{i(N-i)N_i}{N^2} - 1\right] A(T_2)\, A(T_1-T_2) + A(T_2) \quad . \tag{4.43}$$

This can be extended to multiple errors ≥ 2 but the number and complexity of the terms becomes prohibitively large. The major point to note is that, as in the single error case, implementing SA on a general directed tree is poor practice since there is a greater probability of converging to the same signature than in the case of SA on a

Rule	Boundary conditions	Distribution			
		method 1	method 2	method 3	method 4
204	all	Fail	Fail	Fail	Fail
51	all	Pass	Pass	Fail	Fail
60	null	Fail	Fail	Pass	Fail
195	null	Fail	Fail	Pass	Fail
102	null	Fail	Fail	Pass	Fail
153	null	Pass	Pass	Pass	Fail
90	null	Pass	Pass	Fail	Fail
165	null	Pass	Pass	Fail	Fail
150	null	Pass	Pass	Fail	Fail
105	null	Pass	Pass	Fail	Fail
240	cyclic	Pass	Pass	Pass	Fail
15	cyclic	Pass	Pass	Pass	Fail
170	cyclic	Pass	Pass	Pass	Fail
85	cyclic	Pass	Pass	Pass	Fail
150	cyclic	Pass	Pass	Fail	Fail
105	cyclic	Pass	Pass	Fail	Fail
101	cyclic	Pass	Pass	Fail	Fail
154	cyclic	Pass	Pass	Fail	Fail
89	cyclic	Pass	Pass	Fail	Fail
166	cyclic	Pass	Pass	Fail	Fail
75	cyclic	Pass	Pass	Fail	Fail
180	cyclic	Pass	Pass	Fail	Fail
45	cyclic	Pass	Pass	Fail	Fail
210	cyclic	Pass	Pass	Fail	Fail
90, 150 hybrid	null	Pass	Pass	Fail	Fail

Table 4.2: *Equidistribution test results for CA rules of Table 4.1.*

unary tree. For example, if we assume that all branches are unary then Eqn. 4.43 reduces to $\frac{N-1}{N}$. This can be checked against instance i) of the proof since we know that for a unary tree system the signatures will remain different after only one error. Thus, once again we see that the CA rules of Table 4.1 are the most suitable for use in a C-MISR when we consider multiple errors.

A more general, and easily used, measure occurs in Theorem 4.4. It holds equally well for both C-MISRs and L-MISRs provided that all possible inputs are mapped equally over all the possible signatures. This cannot be shown algebraically for most CA rules, so computer simulations were used to show which rules from Table 4.1 satisfied this requirement. The results are summarised in Table 4.2. Notice that most CA rules which implement cyclic groups equally distribute all possible inputs

over all possible signatures. Therefore, we have reduced the number of potential CA rules to be used in a C-MISR using SA methods 1 and 2 to those of Table 4.1. Further analysis of multiple errors beyond that of Theorem 4.8 and Theorem 4.4 is left as an open problem.

It is possible to do empirical studies to verify our estimates on the aliasing probability for the CA rules of Table 4.1. One such test is to check the aliasing probability by inserting errors into a number sequence and checking on the number of error insertions required before an aliased signature occurs (i.e. if 250 error insertions are required then an aliased signature occurred after 250 different error insertions). These results are reported in Table 4.3. It was found that SA methods 1 and 2 did not give quantifiably different results, so only results for method 2 are given. For comparison the results for a L-MISR are also included. We note that both the L-MISR and the C-MISR implementations alias at a rate which is approximately predicted by Theorem 4.4. For example, for a length 9 rule 89 cellular automaton it takes an average of 433 quadruple error insertions before an aliased signature occurs. In addition, no difference can be seen between the performance of the various CA rules of Table 4.1 and the LFSR. Thus, the results of this simulation indicate that the CA rules of Table 4.1 and the LFSR provide essentially equivalent aliasing performance for the above test.

It should be noted that the simulations of Table 4.3 assume error independence. Earlier it was indicated that this is not a realistic assumption since it is known that error outputs from circuits may be dependent. This does not make the results of Table 4.3 irrelevant but merely places them in perspective and also explains some curious results. Some CA rules, such as rules 204 and 51 which maintain or invert the current value respectively, perform well on tests such as shown in Table 4.3, but are entirely unsuitable for SA since it is easy to construct circuits which cannot be tested by these two rules. Other CA rules are similar in that they lead to very regular behaviour and, as for rules 204 and 51, it is possible to construct pathological circuits for these rules. However, as the behaviour of the CA rule becomes more complex, the frequency with which pathological circuits are found decreases. The most complex behaviour is exhibited by CA rule 45 and the rule 90 and 150 hybrid which, as we know from Chapter 2, exhibit pseudorandom behaviour. To indicate why a more pseudorandom MISR circuit is preferable to the L-MISR circuit we note that, in the case of the second LFSR problem circuit of Fig. 4.2(b), the missed error occurred because the signature analyzer cancelled out the error when the LFSR shifted. If a C-MISR using rule 45 or the rule 90 and 150 hybrid had been used, the error would not be cancelled since each bit in the C-MISR is a function of the incoming information and its three neighbour bits. This contrasts with the L-MISR where each bit is a function of the incoming information and just one neighbouring bit. Therefore, it should be expected that a pseudorandom C-MISR such as CA rule 45 would provide better overall SA since it is less prone to dependent errors, as in the example given above.

Using a pseudorandom cellular automaton such as rule 45 for both TPG and MISR makes it possible to implement a testing structure such as in Fig. 4.15. This

		Number of Multiple Errors									
Rule	L	1	2	3	4	5	6	7	8	9	10
LFSR	8	512	211	229	216	261	225	227	243	233	244
204	8	512	237	204	223	217	242	249	207	201	227
51	8	512	209	221	211	218	208	226	242	223	214
60	8	512	210	214	184	235	282	210	207	234	215
195	8	512	241	238	208	207	225	221	254	210	226
102	8	512	234	229	234	233	232	221	211	235	234
153	8	512	251	234	227	205	200	227	230	242	229
90	8	512	191	220	227	213	220	243	222	225	214
150	8	512	214	229	223	200	241	246	227	243	234
105	8	512	220	216	210	235	249	211	203	187	212
165	8	512	211	215	250	235	212	202	218	223	230
240	8	512	239	200	231	228	208	237	185	235	231
15	8	512	205	205	212	210	204	212	199	219	223
170	8	512	202	232	238	201	200	234	207	197	231
85	8	512	222	230	203	247	224	188	231	203	210
LFSR	9	1024	451	445	458	442	462	445	461	483	472
204	9	1024	459	451	385	414	490	458	409	523	542
51	9	1024	482	410	438	410	445	438	410	425	473
60	9	1024	411	458	443	471	480	445	403	500	407
195	9	1024	405	435	511	476	432	469	448	466	443
102	9	1024	417	427	452	412	370	461	434	425	420
153	9	1024	440	466	439	414	427	476	381	494	427
101	9	1024	439	470	435	476	429	439	428	456	434
154	9	1024	460	440	443	472	436	485	444	407	503
89	9	1024	491	430	433	417	505	435	485	402	455
166	9	1024	415	481	434	498	433	451	438	491	395
75	9	1024	472	468	481	482	394	469	440	452	410
180	9	1024	391	451	440	434	411	369	511	449	463
45	9	1024	435	481	488	428	428	468	480	385	487
210	9	1024	429	462	452	467	407	445	457	467	466
90h150	10	2048	856	898	846	863	938	868	863	851	891

Table 4.3: *Performance of CA rules of Table 4.1 for multiple error aliasing using SA methods 1 and 2. Here we consider up to 10 errors in a sequence of 1000 numbers and find the average over 100 different original sequences.*

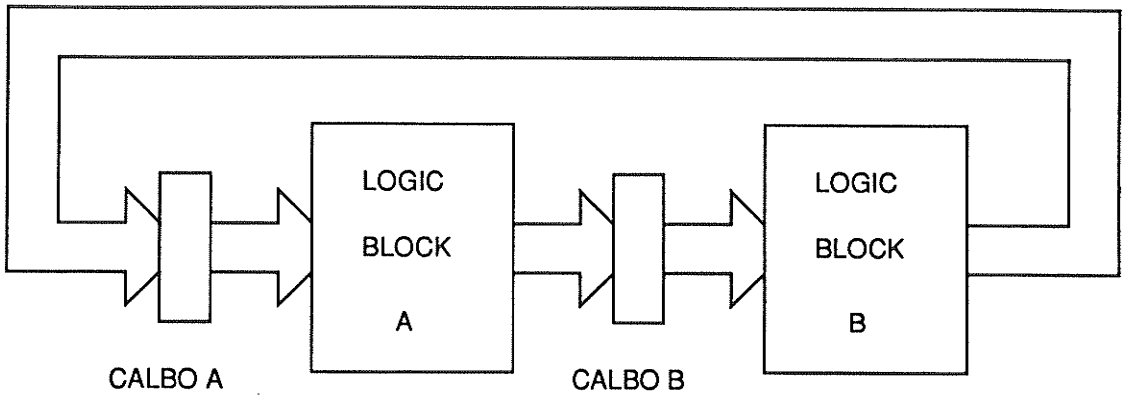


Figure 4.15 : *Using CALBO to test two different logic blocks. During test phase one CALBO A is a C-TPG and CALBO B is a C-MISR and during test phase two CALBO A is a C-MISR and CALBO B is a C-TPG.*

technique is commonly used with BILBO but has, up to now, been impossible with CALBO since the aliasing properties for SA were unknown. Therefore, it would appear that, since the CALBO circuits using methods 1 and 2 proposed in this work perform equally well for SA as traditional LFSR circuits, all applications in which BILBO is used as the BIST methodology can be replaced by a CALBO test structure.

4.7.2.2. Two More C-MISR Techniques

The final methods of signature analysis to be considered here use the techniques shown in Fig. 4.16. Here we use the communication lines of the cellular automaton sites as the points at which the circuit outputs are introduced. This is done by *exclusive-or*ing the passed value from the adjacent cellular automaton site with the corresponding circuit output. Notice that in the technique of Fig. 4.16(top) the gate count is higher than in methods 1 and 2 but if we use the technique of Fig. 4.16(bottom) it is possible to use a $\lceil m/2 \rceil$ site cellular automaton as the C-MISR. These particular C-MISRs can also be described algebraically as follows:

Method 3:

$$a_i(t+1) = \Phi\left[\, a_{i-1}(t) \oplus O_i(t+1),\ a_i(t),\ a_{i+1}(t) \oplus O_i(t+1) \,\right] \qquad (4.44)$$

Method 4:

$$a_i(t+1) = \Phi\left[\, a_{i-1}(t) \oplus O_{2i-1}(t+1),\ a_i(t),\ a_{i+1}(t) \oplus O_{2i}(t+1) \,\right] \qquad (4.45)$$

where

$a_i(t)$ = value of CA site i at time t.
$O_i(t)$ = circuit output from bit i at time t.

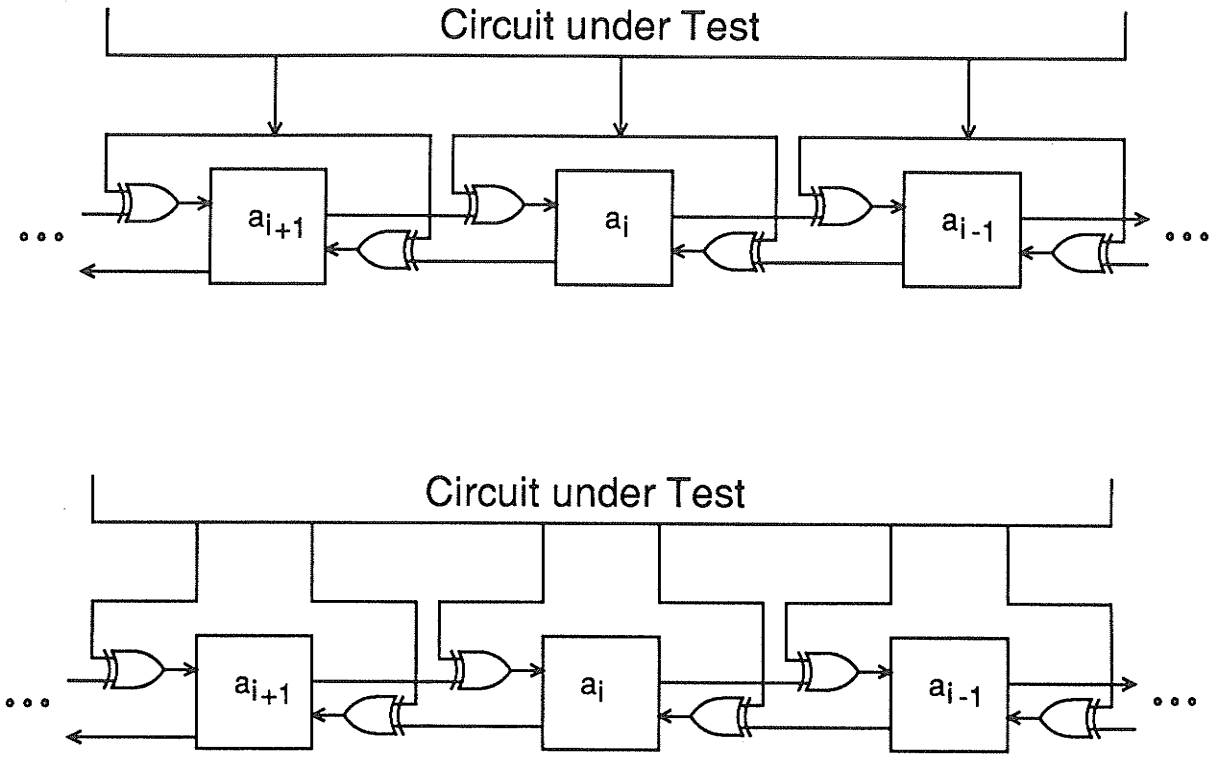


Figure 4.16 : *Two more techniques of SA using CA-based MISRs;*
(top) method 3:

$$a_i(t+1) = \Phi(a_{i-1}(t) \oplus O_i(t+1),\ a_i(t),\ a_{i+1}(t) \oplus O_i(t+1)).$$

(bottom) method 4:

$$a_i(t+1) = \Phi(a_{i-1}(t) \oplus O_{2i-1}(t+1),\ a_i(t),\ a_{i+1}(t) \oplus O_{2i}(t+1)).$$

$\Phi(\ )$ = Particular CA rule of operation implemented.

These two methods give rise to much more complicated behaviour than methods 1 and 2 since methods 3 and 4 create a dynamic hybrid whose rules of operation are based on the current output of the circuit under test. For example, consider CA rule 30 with truth table as given in Table 4.4. We see that, using method 3, if the circuit output to one site is low then that site functions normally. However, if the circuit output is high then the CA rule at that site changes, as shown in Table 4.4, because the communicated values from the left and right neighbours are inverted. This occurs throughout the cellular automaton so we are forming a dynamic two rule hybrid where each site is either CA rule 30 or 210 depending on the corresponding circuit output. For SA method 4 the situation is even more complicated. Consider Table 4.5 where the four possible rules at each site are given. Now a dynamic cellular automaton where any one site could be one of four different rules is created. Another consideration is the boundary conditions. For null boundary conditions we are, in fact, using the most and least significant circuit output values as the boundary inputs of the cellular automaton. This further permutes the overall null boundary behaviour since the

Rule 30							
111	110	101	100	011	010	001	000
0	0	0	1	1	1	1	0
Circuit output = 0 → Rule 30							
111	110	101	100	011	010	001	000
Φ(111)	Φ(110) 0	Φ(101) 0	Φ(100) 0	Φ(011) 1	Φ(010) 1	Φ(001) 1	Φ(000) 1
Circuit output = 1 → Rule 210							
111	110	101	100	011	010	001	000
Φ(010) 1	Φ(011) 1	Φ(000) 0	Φ(001) 1	Φ(110) 0	Φ(111) 0	Φ(100) 1	Φ(101) 0

Table 4.4: *Effect of SA method 3 on a rule 30 C-MISR.*

boundary conditions are now dynamic rather than fixed. The effect is reduced if cyclic boundary conditions are used since the left and right boundary values are normally dynamic.

While analytical study of SA methods 3 and 4 is difficult (less so for method 3) it is possible to repeat the previous empirical studies for methods 1 and 2. We first investigate the distribution of signatures by different rule CALBOs. Computer studies have shown that CA rules other than those of Table 4.1 do not have an equidistribution of signatures using either method 3 or 4. For those rules of Table 4.1 the results in the final two columns of Table 4.2 are presented. Notice that for SA method 3 about half the rules yield equally distributed signatures. However, it is surprising that the two random rules (45 and the 90, 150 hybrid) do result in equally distributed signatures using method 3. For SA method 4 we see that no CA rules result in equally distributed signatures. This would indicate that the general SA theorems proved earlier will not hold for methods 3 and 4 since the signatures are not equally distributed.

It is also possible to study single and multiple error performance using a similar study to that of Table 4.3. In that study it was found that SA methods 1 and 2 yielded essentially equivalent behaviour so one set of results could be used to indicate the performance of both methods. As we have already seen for the distribution of signatures; SA methods 3 and 4 yield very different performance. Therefore, the performance of both methods must be presented separately. In Table 4.6 we see the results for method 3 and in Table 4.7 for method 4.

For method 3 we see that essentially equivalent performance to the LFSR for multiple signatures is possible if a CA rule such as rule 60 with null boundary conditions is used. However, the number of such rules is greatly diminished when compared

Rule 30							
111	110	101	100	011	010	001	000
0	0	0	1	1	1	1	0
Left circuit output = 0 Right circuit output = 0 → Rule 30							
111	110	101	100	011	010	001	000
Φ(111) 0	Φ(110) 0	Φ(101) 0	Φ(100) 1	Φ(011) 1	Φ(010) 1	Φ(001) 1	Φ(000) 0
Left circuit output = 0 Right circuit output = 1 → Rule 45							
111	110	101	100	011	010	001	000
Φ(110) 0	Φ(111) 0	Φ(100) 1	Φ(101) 0	Φ(010) 1	Φ(011) 1	Φ(000) 0	Φ(001) 1
Left circuit output = 1 Right circuit output = 0 → Rule 225							
111	110	101	100	011	010	001	000
Φ(011) 1	Φ(010) 1	Φ(001) 1	Φ(000) 0	Φ(111) 0	Φ(110) 0	Φ(101) 0	Φ(100) 1
Left circuit output = 1 Right circuit output = 1 → Rule 210							
111	110	101	100	011	010	001	000
Φ(010) 1	Φ(011) 1	Φ(000) 0	Φ(001) 1	Φ(110) 0	Φ(111) 0	Φ(100) 1	Φ(101) 0

Table 4.5: *Effect of SA method 4 on a rule 30 C-MISR.*

to those with equivalent LFSR performance when using methods 1 or 2. For the null boundary condition rules we see that either the C-MISR will deliver comparable LFSR performance or it is trivial to alias the C-MISR. However, for cyclic boundary conditions we see that some C-MISRs, such as rule 166, have very poor performance for low error counts but as the total number of errors increases so does aliasing performance. This is not unexpected, but the rate of performance increases more rapidly than might be thought. In any case such performance is not desirable since one must have low aliasing probability at all total error counts. The most puzzling result of SA using method 3 lies in the performance of CA rule 45, a *random rule*. From previous discussions one would expect the aliasing performance of rule 45 to be at least equivalent to that of any other rule but as can be seen it is actually much worse than some rules. The reason for this is left as an open problem.

		Number of Multiple Errors									
Rule	L	1	2	3	4	5	6	7	8	9	10
LFSR	8	512	211	229	216	261	225	227	243	233	244
204	8	1	1	1	1	1	1	1	1	1	1
51	8	1	1	1	1	1	1	1	1	1	1
60	8	512	210	195	219	207	212	224	229	249	230
195	8	512	211	207	250	242	214	217	223	225	226
102	8	512	218	224	236	201	213	183	205	236	260
153	8	512	235	196	221	240	232	216	243	232	181
90	8	1	1	1	1	1	1	1	1	1	1
150	8	1	1	1	1	1	1	1	1	1	1
105	8	1	1	1	1	1	1	1	1	1	1
165	8	1	1	1	1	1	1	1	1	1	1
240	8	512	230	209	235	235	242	221	215	245	230
15	8	512	228	195	215	200	202	244	256	222	238
170	8	512	228	223	220	210	232	198	214	214	228
85	8	512	249	203	227	216	216	220	228	211	194
LFSR	9	1024	451	445	458	442	462	445	461	483	472
204	9	1	1	1	1	1	1	1	1	1	1
51	9	1	1	1	1	1	1	1	1	1	1
60	9	1024	466	422	417	411	541	481	427	391	457
195	9	1024	401	468	451	444	475	464	451	477	429
102	9	1024	479	475	438	424	490	401	424	454	385
153	9	1024	463	441	418	439	421	463	490	433	432
101	9	10	130	238	295	363	386	447	397	322	393
154	9	1	1	1	1	1	1	1	1	1	1
89	9	13	147	260	294	330	378	348	387	382	467
166	9	10	144	188	246	385	284	399	330	448	448
75	9	12	98	268	306	332	407	438	371	400	404
180	9	10	127	256	324	304	356	404	396	448	401
45	9	11	85	237	307	286	304	367	363	391	358
210	9	1	1	1	1	1	1	1	1	1	1
90h150	10	1	1	1	1	1	1	1	1	1	1

Table 4.6: *Performance of CA rules of Table 4.1 for multiple error aliasing using SA method 3. Here we consider up to 10 errors in a sequence of 100 numbers and find the average over 100 different original sequences.*

		Number of Multiple Errors									
Rule	L	1	2	3	4	5	6	7	8	9	10
LFSR	8	512	211	229	216	261	225	227	243	233	244
204	8	1	1	1	1	1	1	1	1	1	1
51	8	1	1	1	1	1	1	1	1	1	1
60	8	18	16	15	15	15	17	17	16	16	15
195	8	16	17	13	16	18	14	14	16	15	15
102	8	16	14	16	18	14	16	19	13	16	15
153	8	18	17	17	18	17	14	15	14	14	16
90	8	14	14	16	17	15	14	15	16	17	17
150	8	16	16	16	14	14	14	15	15	16	16
105	8	16	16	17	15	16	14	19	16	16	17
165	8	16	14	16	16	15	15	14	13	17	15
240	8	19	17	19	17	17	14	15	16	15	17
15	8	19	16	15	15	17	17	14	19	17	15
170	8	14	15	14	15	14	15	16	14	17	14
85	8	18	16	17	18	15	17	20	15	17	15
LFSR	9	1024	451	445	458	442	462	445	461	483	472
204	9	1	1	1	1	1	1	1	1	1	1
51	9	1	1	1	1	1	1	1	1	1	1
60	9	16	12	16	17	17	18	18	16	15	14
195	9	16	15	16	15	13	16	15	15	14	18
102	9	15	15	16	15	16	19	16	16	16	16
153	9	17	15	17	14	15	18	16	17	15	13
101	9	1	1	1	1	1	1	1	1	1	1
154	9	1	1	1	1	1	1	1	1	1	1
89	9	1	1	1	1	1	1	1	1	1	1
166	9	1	1	1	1	1	1	1	1	1	1
75	9	1	1	1	1	1	1	1	1	1	1
180	9	1	1	1	1	1	1	1	1	1	1
45	9	1	1	1	1	2	1	1	1	1	1
210	9	1	1	1	1	1	1	1	1	1	1
90h150	10	15	15	16	14	15	14	15	16	17	14

Table 4.7: *Performance of CA rules of Table 4.1 for multiple error aliasing using SA method 4. Here we consider up to 10 errors in a sequence of 100 numbers and find the average over 100 different original sequences.*

For method 4 we see that all CA rules yield very poor performance which appears to be independent of the number of errors in the circuit output sequence. Therefore, none of the investigated rules is suitable for SA using method 4.

4.8. IMPLEMENTATION CONSIDERATIONS

A major advantage, and perhaps the most important, of using cellular automata over LFSRs, is the ease with which the length, or number of outputs, of the TPG can be changed. For example, consider the following scenario [Rosen1987].

> *Suppose that a design will have N inputs and that the designers balk at having full N+N sites in their xxLBO. The experience summarised in [Waicukauski1987], [Barzilai1983] suggests that rather few of the faults will be seriously affected by correlation, and hence that rather few of the N inputs will need neighbouring dummies. Squeezing a N+(few) is easier than squeezing N+N. But we won't know how many dummy sites are really wanted until late in the game, after extensive fault simulation. This is an awkward time to consult a list of primitive polynomials, find out that the list can't get N+3 without going to N+30 and a lot more taps, cajole an algebraist into helping out, and so on.*

On the other hand, in a CA-based implementation there is much less cross-correlation so dummy sites will probably not be needed. However, if the length must be increased or decreased because of needed dummy sites, or a change in the number of inputs in the circuit under test has been made, a CA approach merely requires changing the number of cells (remember each cell is the same) and using a new starting value. This contrasts with the LFSR which requires complete redesign even if only one site is added. Therefore, a C-TPG, or C-MISR, generally may be of any convenient length and thereby imposes few restrictions on the number of inputs of the circuit to be tested. This contrasts with L-TPGs and L-MISRs where the desire to avoid redesigning the test circuit may force unnecessarily harsh restrictions on the number of inputs or outputs from a circuit which must incorporate BIST.

Figure 4.17 illustrates the topology of both BILBO and a rule 30 based CALBO circuit. As indicated in Fig. 4.17(top), for a 4 cell BILBO operating in the LFSR mode, outputs Q_4, Q_3, and Q_1 are tapped and fed back through *exclusive-or* gates to the MUX input. One immediate difficulty lies in selecting the appropriate feedback taps; that is, the taps are not independent of n for maximum length polynomial division. Another potential difficulty arises in having to send the higher order tap back to the MUX input, as this distance may be a significant fraction of the chip dimension and grows (linearly) with n, the number of cells. A technological fix may include the use of larger *exclusive-or* gates, hierarchical drivers, or wider and thicker metal lines to circumvent problems associated with delay and current density [Mead1980], [Card1987a]. The cost involved with this approach arises from a reduced topological regularity of the basic cells, a major design deterrent for increasing n. In any event, a

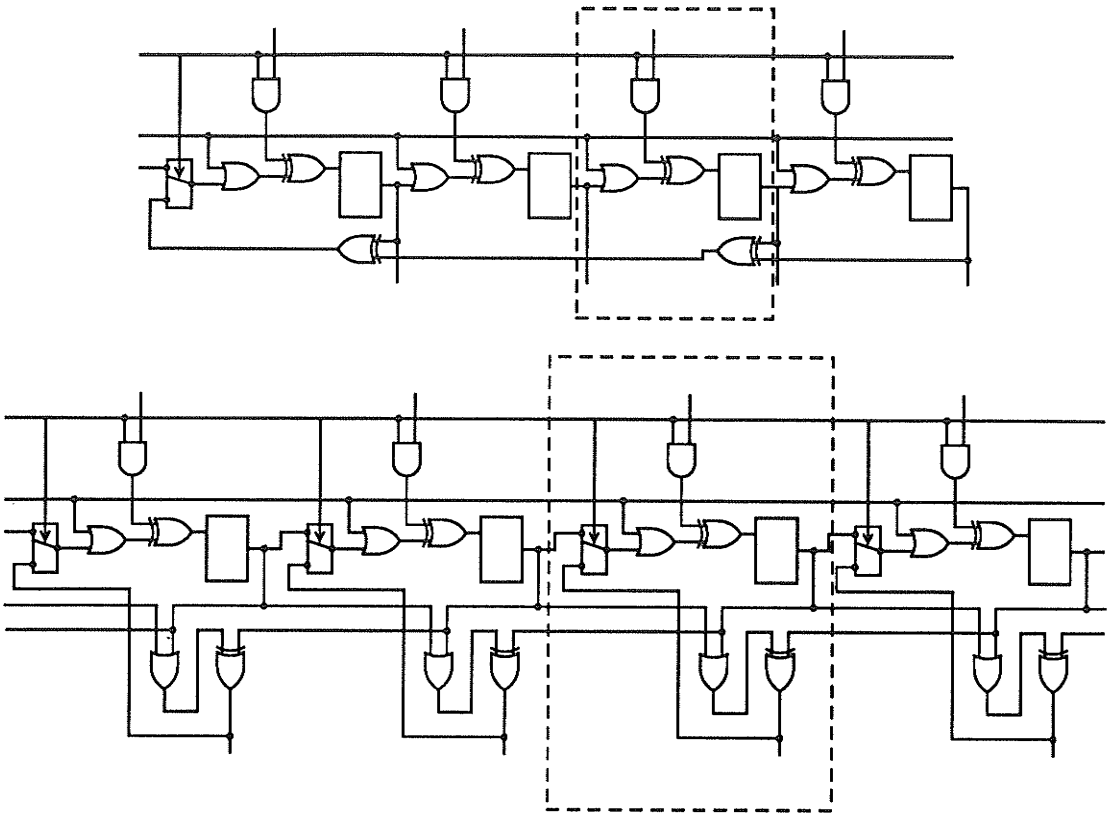


Figure 4.17 : *Topology associated with (top) a 4 bit BILBO and (bottom) a 4 bit rule 45 CALBO test circuit. Note the use of a 2-1 multiplexer at the input to each CALBO stage. Rule 45 is implemented using the q' output of the flipflop.*

time penalty due to propagation delays is experienced of $\Omega(log\ n)$, or even worse, of $\Omega(n)$ if current density limitations [Card1987a] are taken into account. As indicated in Fig. 4.17(bottom), a CALBO circuit does not suffer from this symptom as the required communication can be restricted to nearest neighbours, and if null boundary conditions are suitable, then no feedback is required. In addition, even if feedback is required (cyclic boundary conditions) no gate delay is experienced as opposed to the LFSR where the feedback path has at least one gate delay. On the other hand, the hardware of the basic cell has been increased to accommodate the required storage of the present state, the local logic to implement a given CA rule, and the incorporation of transmission-gate multiplexers.

One area of further investigation is warranted; the system size n at which the utility of the CALBO approach is expected to improve upon that of the BILBO test circuitry must be examined. In comparing silicon implementations of CALBO with BILBO, we suggest the adoption of an established criterion such as the AT (area-time) metric for VLSI [Thompson1980]. The constant factors in A and T are important in non-asymptotic design decisions, and these factors are easily accounted for in a relative

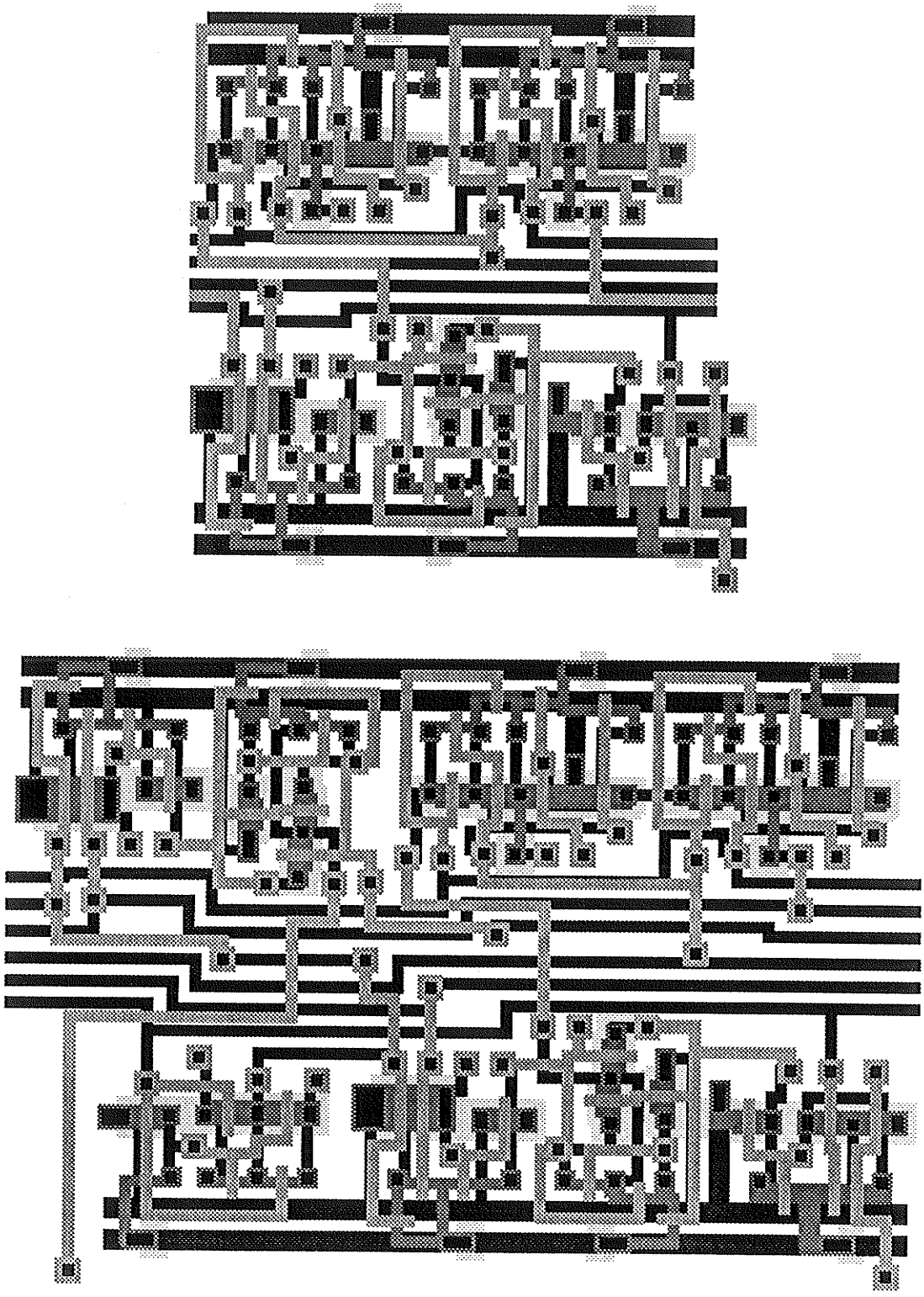


Figure 4.18 : *Static CMOS implementations for units cells of: (top) the BILBO circuit. (bottom) the CALBO circuit.*

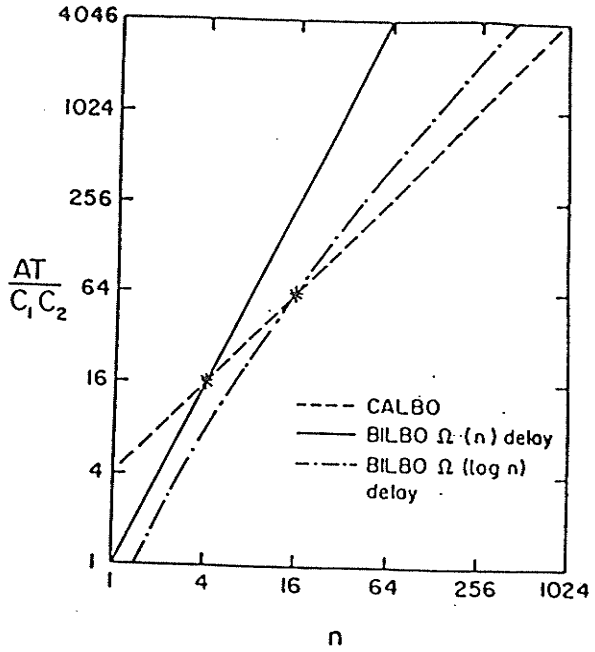


Figure 4.19 : *AT comparison of BILBO and CALBO versus register length, n.*

manner (i.e. it is easy to compare absolute AT metrics of CALBO and BILBO for a given technology, where constant factors are related). In addition, we suggest the use of a heuristic based upon wiring difficulty and technological fixes. From Fig. 4.18 we see the unit cell for static CMOS implementations of a CALBO and BILBO circuit using custom CMOS cells. Notice that the CALBO circuit requires at most two times the area per cell as the unit cell for the BILBO.[4.8] This factor of two accounts for the logic required to perform the cellular automaton operation (XOR, OR) at each cell along the array, as well as for the 2×1 multiplexer which allows each cell to feed back to itself. A d flipflop is used in the array, since a one bit memory is necessary for the operation of the cellular automaton. Alternative latches may also be employed as in Scan Set or LSSD. The increased local wiring complexity associated with CALBO is traded against the increased global wiring complexity of BILBO. The area for BILBO increases as $C_1 n$, with n the number of register cells, whereas the area for CALBO increases approximately as $2C_1 n$. This ignores the area of the additional wiring in BILBO, which will be problematic for sufficiently large n. A time complexity comparison favours CALBO since communication is restricted to nearest neighbours, whereas communication in BILBO in the general case may extend over a considerable fraction of the chip width. In the hierarchical driver scheme one assumes this time to increase

[4.8] The CMOS building blocks are taken from the University of Manitoba 3μm CMOS cell library [Card1987b] and were constructed using the ELECTRIC IC design tool [Rubin1983]. The exact ratio of the BILBO and CALBO circuits is 1.8, ignoring the added area required for feedback. Therefore, since the area of the BILBO feedback circuit is much larger than that of the CALBO we can say the factor of 2 is a conservative estimate favouring the LFSR.

as $C_2 \log n$ for BILBO; it is simply C_2 for CALBO. The AT metric implies that the CALBO circuit approach is preferred to the BILBO circuit when $n \geq 4$.[4.9]

If the delay associated with communication in BILBO were dominated by current density limitations, the time factor would become $C_2 n$ for BILBO. As a consequence, the AT metric shown in Fig. 4.19 would imply that the CALBO circuit approach would always be preferred to the BILBO circuit for lengths ≥ 4. In any event, the virtue of a BIST technique, such as BILBO or CALBO, would have to be questioned for a small number of control points, as we are attempting to exploit the pseudorandom number capabilities of such circuits. Thus, for the reasons discussed above, when n is sufficiently large that statistical tests are the only practical means available to test a combinational circuit, the CALBO circuit described here provides a very attractive alternative to BILBO.

4.9. CONCLUSIONS

In summary, an alternative BIST technique has been presented which may find application as an alternative to BILBO or similar schemes in design for testability. Advantages arise from reduced cross-correlation between the individual bit streams in the CALBO case, and hence, an increased statistical independence in the set of test vectors. The only required effort, once the appropriate CA rule has been selected, is to determine a suitable starting value or seed. This compares to the selection of taps in the LFSR as the length of the array is varied. A major advantage of CA-based test pattern generators versus those based on LFSRs is the suitability of cellular automata for use in CAD tools. The irregular location of feedback taps in the LFSR requires a complete redesign of the LFSR when the physical length is changed, whereas for cellular automata a change in length merely requires adding or removing an appropriate number of cellular automaton cells.

Fault coverage estimates which have been developed for LFSR-based pseudorandom testing have been shown to more applicable to the CA-based schemes proposed here. Therefore, much of the current knowledge with respect to the fault coverage in a random testing can be readily used in a CA-based pseudorandom test environment.

Signature analysis techniques for cellular automata have been examined and were shown to provide equivalent aliasing performance for equally likely errors to the LFSR, with restrictions on the particular CA rule of operation used. For the more demanding and realistic case of dependent errors the improved pseudorandomness of the CA-based signature circuits was shown to provide better aliasing performance than the LFSR for certain circuit faults. Finally, with respect to the circuit area used by cellular automata and the LFSR. It was shown that CA structures use equivalent area

[4.9] AT for BILBO and CALBO are equivalent when $C_1 C_2 n \log n = 2 C_1 C_2 n;\ n = 2^2 = 4$.

within a factor of two and show much improved speed of operation as compared to the LFSR.

Further effort is required in exploring the wide range of applications for pseudorandom cellular automata in VLSI testing. Among these applications, their suitability for pseudorandom weighted pattern generation requires further investigation.

Chapter 5
Conclusions and Suggestions for Further Study

5.1. SUMMARY AND CONCLUSIONS

In this thesis we have been concerned with the study of parallel VLSI systems for nondeterministic algorithms. We have especially emphasised the requirements of pseudorandom number generation for such systems. In this regard we have seen that conventional techniques of pseudorandom number generation, both software and hardware, are inadequate for parallel pseudorandom number generation since the silicon area required is either much too large per bit of random number or the approximation of random behaviour is inadequate. A new pseudorandom number generator based on simple one-dimensional class 3 cellular automata was proposed which provides improved area and time properties. These new generators were extensively studied for randomness properties and cycle lengths. It appears that the rule 90 and 150 hybrid exhibits the best area-time efficiency although the degree of randomness may be less than that of CA rule 30 or 45. We have observed that CA rule 30 provides the highest quality randomness but with the smallest cycle lengths. However, for long registers, concerns as to the length of the cycle are somewhat tempered since more than adequate cycle length can be obtained.

Two models from statistical mechanics, the percolation and Ising models, have been studied and fine-grained parallel architectures for their Monte Carlo simulation were proposed. The resulting architectures would be used as coprocessors, or hardware experts, to a host computer in order to attack the computationally intensive problem of updating the lattice sites of each model. This results in a very efficient architecture on which lattice updates are made at rates several orders of magnitude faster than on other computing systems. These high speed architectures allow the possibility of more exact Monte Carlo simulation especially at, or near, the critical point of the phase transition. Correctness of these architectures was verified by simulation to extract the correct critical exponents for both models.

A further application of the area-time efficient pseudorandom number generator developed for this work is in pseudorandom testing of digital circuits. It is shown that the proposed CALBO approach offers improved fault detection versus the traditional BILBO circuit. In addition, a study of the signature analysis properties of cellular automata shows that equivalent aliasing performance to the LFSR can be achieved by

using the cyclic group rules such as CA rule 45 and the rule 90 and 150 hybrid.

5.2. SUGGESTIONS FOR FURTHER WORK

This investigation has shown that it is feasible to consider implementing VLSI solutions to nondeterministic algorithms. However, there are still a number of problems which have arisen from this work.

1. A study should be carried out into the reasons for the inexplicable behaviour of the rule 90 and 150 hybrid, especially with regard to its site and time spacing behaviour. In addition, other potential hybrids should be examined for possible use as pseudorandom number generators.
2. The employment of two-dimensional cellular automata and those of higher dimensions should be studied for use in parallel architectures such as those discussed here. For example, considerable potential improvement in the architectures of Chapter 3 exists if a two-dimensional cellular automaton were used as the pseudorandom number generator.
3. The proposed architectures for the percolation and Ising models should be implemented in custom VLSI and studied for circuit improvements. In addition, implementation specific problems such as interchip communication and on-chip I/O problems not discussed in this work should be addressed.
4. The proposed Ising model architecture based on the mapping of Domany and Kinzel warrants further study and simulation to further verify its appropriateness for Ising model computations.
5. The fundamental processor architecture of Chapter 3 should be extended to handle other statistical mechanical problems such as the Heisenburg model, growth models, and spin glasses.
6. A complete fault simulation study of the proposed test pattern generator would be useful especially with regards to examining stuck-open fault detectability.
7. The possibility of weighted test pattern generation should be examined especially from the perspective of weight selectivity.
8. Other design for testability structures using cellular automata are possible.
9. Simulation of cellular automata-based signature analysers should be made using real circuits rather than a random input model.

5.3. CONCLUDING REMARKS

This work has shown the great potential for VLSI solution of nondeterministic algorithms. The resulting systems have demonstrated a great improvement over conventional solutions. The eventual employment of the proposed systems depends on whether the need for these efficient solutions can justify the associated development costs.

References

1. **[Ando1980]** H. Ando, "Testing VLSI with Random-Access Scan," *Compcon 80*, pp. 50-52, San Francisco, Feb. 1980.

2. **[Barber1985]** M.N. Barber, R.B. Pearson, D. Toussaint, and J.L. Richardson, "Finite-Size Scaling in the Three-Dimensional Ising Model," *Phys. Rev. B*, vol. 32, No. 3, pp. 1720-1730, Aug. 1985.

3. **[Barzilai1983]** Z. Barzilai and B.K. Rosen, "Comparison of AC Self-Testing Procedures," *Proceedings of the 1983 IEEE International Test Conference*, pp. 89-94, IEEE Computer Society, Silver Spring, MD, 1983.

4. **[Baschiera1984]** D. Baschiera and B. Courtois, "Testing CMOS: A Challenge," *VLSI Design*, pp. 58-62, Oct. 1984.

5. **[Bate1987]** J.A. Bate and D.M. Miller, "The Exhaustive Testing of Stuck-open Faults in CMOS Combinational Circuits," in *Developments in Integrated Circuit Testing*, ed. D.M. Miller, Academic Press, NewYork, NewYork, 1987.

6. **[Benowitz1975]** N. Benowitz, D.F. Calhoun, G.E. Alderson, J.E. Baver, and C.T. Joeckel, "An Advanced Fault Isolation System for Digital Logic," *IEEE Transactions on Computers*, vol. C-24, No. 5, pp. 489-497, May 1975.

7. **[Berlekamp1982]** E.R. Berlekamp, J.H. Conway, and R.K. Guy, *Winning Ways for your Mathematical Plays,* Academic Press, NewYork, NewYork, 1982.

8. **[Bhavsar1981]** D.K. Bhavsar and R.K. Heckelman, "Self Testing by Polynomial Division," *Proc. 1981 International Test Conference*, pp. 208-216, 1981.

9. **[Binder1975]** K. Binder, "Monte Carlo Computer Experiments on Critical Phenomena and Metastable States," *Advances in Physics*, pp. 917-939, 1975.

10. **[Binder1979]** K. Binder, *Monte Carlo Methods in Statistical Physics,* Springer-Verlag, New York, New York, 1979.

11. **[Binder1980]** K. Binder and D.P. Landau, "Phase Diagrams and Critical Behaviour in Ising Square Lattices with Nearest- and Next-Nearest-Neighbour Interactions," *Phys. Rev. B*, vol. 21, No. 5, pp. 1941-1962, March 1980.

12. **[Binder1984]** K. Binder, *Applications of the Monte Carlo Method in Statistical Physics,* Springer-Verlag, New York, New York, 1984.

13. **[Box1958]** G.E.P. Box, M.E. Muller, and G. Marsaglia, *Annuls of Math. Stat.*, vol. 28, p. 610, 1958.

14. **[Brglez1985]** F. Brglez, P. Pownall, and R. Hum, "Accelerated TPG and Fault Grading via Testability Analysis," *Proc. IEEE International Symp. Circuits and Systems (ISCAS)*, pp. 695-698, Kyoto, Japan, June 1985.

15. **[Broadbent1957]** S.R. Broadbent and J.M. Hammersley, "Percolation Processes. I. Crystals and Mazes," *Proc. Camb. Phil. Soc.*, vol. 53, p. 629, 1957.

16. **[Burks1970]** A. W. Burks, *Essays on Cellular Automata,* Univ. of Illinois Press, Urbana, Ill., 1970.

17. **[Card1987a]** H.C. Card, P.G. Gulak, R.D. McLeod, and W. Pries, "λT Complexity Measures for VLSI Computations in Constant Chip Area," *IEEE Trans. on Computers*, vol. C-36, pp. 112-117, Jan 1987.

18. **[Card1987b]** H.C. Card, R.D. McLeod, and P.D. Hortensius, "Custom Chips for Signal Processing using a University CMOS Cell Library," *Proc. 7th Intl. Conf. on Custom and Semicustom ICs*, London, England, Nov 3-5, 1987.

19. **[Carter1982]** J.L. Carter, "The Theory of Signature Testing for VLSI," *Proc. 14th Ann. ACM. Symp. Theory of Computing*, pp. 66-76, 1982.

20. **[Chan1977]** A.Y. Chan, "Easy-to-Use Signature Analyzer Accurately Troubleshoots Complex Logic Circuits," *Hewlett-Packard Journal*, pp. 9-14, May 1977.

21. **[Chin1984]** C.K. Chin and E.J. McCluskey, "Weighted Pattern Generation for Built-In Self-Test," *Center for Reliable Computing Tech Rep. No. 84-7*, Standford University, Standford, CA, 1984.

22. **[Chin1987]** C.K. Chin and E.J. McCluskey, "Test Length for Pseudorandom Testing," *IEEE Transactions on Computers*, vol. C-36, No. 2, pp. 252-256, Feb. 1987.

23. **[Codd1968]** E. F. Codd, *Cellular Automata,* Academic Press, New York, New York, 1968.

24. **[Coveyou1967]** R.R. Coveyou and R.D. MacPherson, "Fourier Analysis of Uniform Random Number Generators," *Journal of the ACM*, vol. 14, pp. 100-119, 1967.

25. **[Daniels1985]** R.G. Daniels and W.C. Bruce, "Built-In Self-Test Trends in Motorola Microprocessors," *IEEE Design and Test*, pp. 64-71, April 1985.

26. **[Darema1987a]** F. Darema, S. Kirkpatrick, and V.A. Norton, "Parallel Algorithms for Chip Placement by Simulated Annealing," *IBM Journal of Res. Develop.*, vol. 31, No. 3, p. 391, May 1987.

27. **[Darema1987b]** F. Darema and G.F. Pfister, "Multiprocessor parallelism for VLSI CAD on the RP3," *IEEE Design and Test of Computers*, vol. 4, No. 5, Oct. 1987.

28. **[Denyer1985]** P. Denyer and D. Renshaw, *VLSI Signal Processing,* Addison-Wesley, Reading, Mass., 1985.

29. **[Domany1984]** E. Domany and W. Kinzel, "Equivalence of Cellular Automata to Ising Models and Directed Percolation," *Phys. Rev. Lett.*, vol. 53, pp. 311-314, 1984.

30. **[Eichelberger1977]** E.B. Eichelberger and T.W. Williams, "A Logic Design Structure for LSI Testing," *Proc. 14th Design Automation Conf.*, pp. 462-468, New Orleans, June 1977.

31. **[Eldred1959]** R.D. Eldred, "Test Routines Based on Symbolic Logical Statements," *Journal of the ACM*, vol. 6, No. 1,, pp. 33-36, 1959.

32. **[Enting1977]** I.G. Enting, "Crystal Growth Models and Ising Models: Disorder Points," *Journal Phys. C*, vol. 10, p. 1379, 1977.

33. **[Enting1978]** I.G. Enting, "Crystal Growth Models and Ising Models IV: Graphical Solution by Correlations," *J. Phys. A*, vol. 11, No. 3, pp. 555-562, 1978.

34. **[Essam1972]** J.W. Essam, "Percolation and Cluster Size," in *Phase Transitions and Critical Phenomena*, ed. C. Domb, M.S. Green, vol. 2, p. 197, Academic Press, New York, New York, 1972.

35. **[Essam1978]** J.W. Essam, D.S. Gaunt, and A.J. Guttmann, "Percolation Theory at the Critical Dimension," *Journal of Phys. A*, vol. 11, p. 1983, 1978.

36. **[Ferdinand1969]** A.E. Ferdinand and M.E. Fisher, "Bounded and Inhomogeneous Ising Models. I. Specific Heat Anaomaly of a Finite Lattice," *Phys. Rev.*, vol. 185, p. 832, 1969.

37. **[Fisher1971]** M.E. Fisher, in *Critical Phenomena, Proceedings of the International School of Physics, "Enrico Fermi Course 51*, ed. M.S. Green, p. 73, Academic Press, New York, New York, 1971.

38. **[Fishman1982]** G.S. Fishman and L.R. Moore, "A Statistical Evaluation of Multiplicative Congruential Random Number Generators with Modulus 2^{31} - 1," *Journal of the American Statistical Association*, vol. 77, No. 377, pp. 129-136, March 1982.

39. **[Frisch1963]** H.L. Frisch and J.M. Hammersley, "Percolation Processes and Related Topics," *Journal Soc. Indust. Appl. Math*, vol. 11, pp. 894-918, 1963.

40. **[Frohwerk1977]** R.A. Frohwerk, "Signature Analysis: A New Digital Field Service Method," *Hewlett Packard Journal*, pp. 2-8, May 1977.

41. **[Fujiwara1985]** H. Fujiwara, *Logic Testing and Design for Testability,* The MIT Press, Cambridge, Mass., 1985.

42. **[Funatsu1975]** S. Funatsu, N. Wakatsuki, and T. Arima, "Test Generation Systems in Japan," *Proc. 12th Design Automation Conf.*, pp. 114-122, Boston, June 1975.

43. **[Gardner1971]** M. Gardner, "Mathematical Games," *Sci. Amer.*, vol. 224, 1971.

44. **[Garey1979]** M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness,* Freeman, San Francisco, California, 1979.

45. **[Golomb1982]** S.W. Golomb, *Shift Register Sequences,* Holden-Day, San Francisco California, 1982.

46. **[Grossmann1977]** S. Grossmann and S. Thomae, "Invariant Distributions and Stationary Correlation Functions of One-Dimensional Discrete Processes," *Z. Naturforsch A*, vol. 32, pp. 1353-1363, 1977.

47. **[Harris1960]** B. Harris, "Probability Distributions Related to Random Mappings," *Ann. Math. Stat.*, vol. 31, p. 1045, 1960.

48. **[HP1978]** Hewlett-Packard, "A Designers Guide to Signature Analysis," *Application Note*, p. 222, 1978.

49. **[Hoogland1983]** A. Hoogland, J. Spaa, B. Selman, and A. Compagner, "A Special Purpose Processor for the Monte Carlo Simulation of Ising Spin Systems," *J. Comput. Phys.*, vol. 51, p. 250, 1983.

50. **[Hoshen1976]** J. Hoshen and R. Kopelman, "Percolation and Cluster Distribution. I. Cluster Multiple Labelling Technique and Critical Concentration," *Phys. Rev.*, vol. 14, p. 3428, 1976.

51. **[Ising1925]** E. Ising, "Beitrag zur Theorie des Ferromagnetismus (A Contribution to the Theory of Ferromagnetism)," *Z. Physik*, vol. 31, p. 253, 1925.

52. **[Kendal1938]** M.G. Kendall and B.B. Smith, "Randomness and Random Sampling Numbers," *Journal of the Royal Statistical Soc.*, vol. 101, pp. 162-164, 1938.

53. **[Kinzel1985]** W. Kinzel, "Phase Transitions of Cellular Automata," *J. Phys.*, vol. B58, p. 229, 1985.

54. **[Kirkpatrick1973]** S. Kirkpatrick, "Percolation and Conduction," *Rev. Mod. Phys.*, vol. 45, p. 574, 1973.

55. **[Kirkpatrick1977]** S. Kirkpatrick, "Percolation Thresholds in Ising Magnets and Conducting Mixtures," *Phys. Rev. B*, vol. 15, pp. 1533-1538, 1977.

56. **[Kirkpatrick1981]** S. Kirkpatrick and E.P. Stoll, "A Very Fast Shift-Register Sequence Random Number Generator," *Journal of Computational Physics*, vol. 40, pp. 517-526, 1981.

57. **[Kirkpatrick1983]** S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi, "Optimization by Simulated Annealing," *Science*, vol. 220, No. 4598, pp. 671-680, May 1983.

58. **[Kirkpatrick1978]** S. Kirkpatrick, "Models of Disordered Materials," in *Ill Condensed Matter*, ed. R. Balian, R. Maynard, G. Toulouse, pp. 323-403, World Scientific, Singapore, 1983.

59. **[Kirkpatrick1985]** S. Kirkpatrick and R.H. Swendsen, "Statistical Mechanics and Disordered Systems," *Commun. ACM*, vol. 28, pp. 363-373, 1985.

60. **[Klein1978]** W. Klein, H.E. Stanley, P.J. Reynolds, and A. Congilo, "Renormalization-Group Approach to the Percolation Properties of the Triangular Ising Model," *Phys. Rev. Letters*, vol. 41, p. 1145, 1978.

61. **[Knuth1981]** D. Knuth, *Seminumerical Algorithms,* Addison Wesley, Reading, Mass., 1981.

62. **[Konemann1979]** B. Konemann, J. Mucha, and G. Zwiehoff, "Built-in Logic Block Observation Techniques," *Proc. 1979 International Test Conf.*, pp. 37-41, Cherry Hill, N.J., Oct. 1979.

63. **[Kuban1984]** J.R. Kuban and W.C. Bruce, "Self-Testing the Motorola MC6804P2," *IEEE Design and Test*, vol. 1, No. 2, pp. 33-41, May 1984.

64. **[Kung1980]** H. T. Kung and C. E. Leiserson, in *Introduction to VLSI Systems*, pp. 271-292, Addison-Wesley, Reading, Mass., 1980.

65. **[Kung1985]** S. Y. Kung, H. J. Whitehouse, and T. Kailath, *VLSI and Modern Signal Processing,* Prentice-Hall, Englewood Cliffs, New Jersey, 1985.

66. **[Landau1976a]** D.P. Landau, "Finite-Size Behaviour of the Ising Square Lattice," *Phys. Rev. B*, vol. 13, No. 7, pp. 2997-3010, April 1976.

67. **[Landau1976b]** D.P. Landau, "Finite-Size Behaviour of the Simple-Cubic Ising Lattice," *Phys. Rev. B*, vol. 14, No. 1, pp. 255-262, July 1976.

68. **[Landau1980]** D.P. Landau, "Phase Transitions in the Ising Square Lattice with Next-Nearest-Neighbour Interactions," *Phys. Rev. B*, vol. 21, No. 3, pp. 1285-1297, Feb. 1980.

69. **[Landau1968]** L.D. Landau and E.M. Lifshitz, in *Statistical Physics*, Pergamon Press, London, England, 1968.

70. **[Lenz1920]** Lenz, "Magnetic Phenomena in Paramagnetic Salts," *Z. Physik*, vol. 21, pp. 613-615, 1920.

71. **[Letham1986]** L. Letham, D. Hoff, and A. Folmsbee, "A 128K EPROM Using Encryption of Pseudorandom Numbers to Enable Read Access," *IEEE J. Solid-State Circuits*, vol. SC-21, pp. 881-888, 1986.

72. **[Lewis1969]** P.A.W. Lewis, A.S. Goodman, and J.M. Miller, "A Pseudorandom Number Generator for the IBM 360," *IBM Systems Journal*, vol. 8, No. 2, pp. 136-146, 1969.

73. **[Lewis1973]** T.G. Lewis and W.H. Payne, "Generalized Feedback Shift Register Pseudorandom Number Algorithm," *J. Assoc. Comput. Mach.*, vol. 20, p. 456, 1973.

74. **[Lin1983]** S. Lin and D.J. Costello, *Error Control Coding: Fundamentals and Applications,* Prentice-Hall, Englewoods Cliffs, New Jersey, 1983.

75. **[Losq1976]** J. Losq, "Efficiency of Random Compact Testing," *IEEE Transactions on Computers*, vol. C-27, No. 6, pp. 613-620, June 1976.

76. **[MacLaren1965]** M.G. MacLaren and G. Marsaglia, "Uniform Random Number Generators," *Journal of the ACM*, vol. 12, No. 1, pp. 83-89, 1965.

77. **[Malaiya1984]** Y.K. Malaiya and S. Yang, "The Coverage Problem for Random Testing," *Proc. of the 1984 International Test Conf.*, pp. 237-245, Nov. 1984.

78. **[Margolina1983]** A. Margolina, Z. Djordjervic, D. Stauffer, and H.E. Stanley, "Corrections to Scaling for Branched Polymers and Gels," *Phys. Rev.*, vol. B28, p. 1652, 1983.

79. **[Marsaglia1968]** G. Marsaglia, "Random Numbers Fall Mainly on the Planes," *Proc. Nat. Acad. Science*, vol. 61, No. 1, pp. 25-28, Sept. 1968.

80. **[Marsaglia1972]** G. Marsaglia, "The Structure of Linear Congruential Sequences," in *Application of Number Theory to Numerical Analysis*, ed. S.K. Zaremba, Academic Press, NewYork, NewYork, 1972.

81. **[Marsaglia1984]** G. Marsaglia, "A Current View of Random Number Generators," *Proc. Computer Science and Statistics, 16th Symposium on the Interface*, Atlanta, Georgia, March 1984.

82. **[Martin1984]** O. Martin, A. Odlyzko, and S. Wolfram, "Algebraic Properties of Cellular Automata," *Comm. Math. Phys.*, vol. 93, p. 219, 1984.

83. **[May1976]** R.M. May, "Simple Mathematical Models with very Complicated Dynamics," *Nature*, vol. 261, pp. 459-467, 1976.

84. **[McCluskey1985a]** E.J. McCluskey, "Built-In Self-Test Techniques," *IEEE Design and Test*, pp. 21-28, April 1985.

85. **[McCluskey1985b]** E.J. McCluskey, "Built-In Self-Test Structures," *IEEE Design and Test*, pp. 29-36, April 1985.

86. **[McCoy1973]** B.M. McCoy and T.T. Wu, *The Two-Dimensional Ising Model,* Harvard University Press, Cambridge, Mass., 1973.

87. **[McGonigal1987]** G.C. McGonigal and M.I. Elmasry, "Generation of Noise by Electronic Iteration of the Logistic Map," *IEEE Trans. on Circuits and Systems*, accepted for publication 1987.

88. **[Mead1980]** C.A. Mead and L. Conway, *Introduction to VLSI Systems,* Addison Wesley, Reading, Mass., 1980.

89. **[Meggitt1961]** J.E. Meggitt, "Error Correcting Codes and Their Implementation," *IRE Trans. on Inform. Theory*, vol. IT-17, pp. 232-244, Oct. 1961.

90. **[Metropolis1953]** N. Metropolis, A.W. Rosenbluth, M.N. Rosenbluth, A.H. Teller, and E. Teller, "Equation of State Calculations by Fast Computing Machines," *J. Chem. Phys.*, vol. 21, p. 1087, 1953.

91. **[Miller1987]** D.M. Miller, *Developments in Integrated Circuit Testing,* Academic Press, NewYork, NewYork, 1987.

92. **[Mucha1986]** J.P. Mucha, W. Daehn, and J. Gross, "Self-Test in a Standard Cell Environment," *IEEE Design and Test*, vol. 3, No. 6, pp. 35-41, Dec. 1986.

93. **[Muzio1987]** J.C. Muzio, F. Ruskey, R.C. Aitken, and M. Serra, "Aliasing Probabilities of Some Data Compression Techniques," in *Developments in Integrated Circuit Testing*, ed. D.M. Miller, Academic Press, NewYork, NewYork, 1987.

94. **[vonNeumann1963]** J. von Neumann, "The General and Logical Theory of Automata," in *J. von Neumann Collected Works* , ed. A.H. Taub, p. 288, 1963.

95. **[Niemeyer1974]** T. Niemeyer and J.M.J. van Leeuwen, "Wilson Theory for 2-Dimensional Ising Spin Systems," *Physica*, vol. 71, p. 17, 1974.

96. **[Onsager1944]** L. Onsager, "Crystal Statistics. I. A Two-Dimensional Model with an Order-Disorder Transition," *Phys. Rev.*, vol. 65, p. 117, 1944.

97. **[Packard1985a]** N.H. Packard, "Complexity of Growing Patterns in Cellular Automata," in *Dynamical Systems and Cellular Automata*, ed. J. Demonget, E. Goles, and M. Tchuente, Academic Press, NewYork, NewYork, 1985.

98. **[Packard1985b]** N.H. Packard and S. Wolfram, "Two-Dimensional Cellular Automata," *J. Stat. Phys.*, vol. 38, pp. 901-946, 1985.

99. **[Papoulis1965]** A. Papoulis, *Probability, Random Variables, and Stochastic Processes,* McGraw-Hill, NewYork, NewYork, 1965.

100. **[Parisi1985]** G. Parisi and F. Rapuano, "Effects of the Random Number Generator on Computer Simulations," *Physics Letters*, vol. 157B, No. 4, pp. 301-302, July 18, 1985.

101. **[Pawley1984]** G.S. Pawley, R.H. Swendsen, D.J. Wallace, and K.G. Wilson, "Monte Carlo Renormalization-Group Calculations of Critical Behaviour in the Simple-Cubic Ising Model," *Phys. Rev. Lett.*, vol. B29, p. 4030, 1984.

102. **[Payne1971]** W.H. Payne and T.G. Lewis, "Conditional Bit Sampling: Accuracy and Speed," in *Mathematical Software*, ed. J.R. Rice, pp. 331-345, Academic Press, New York, New York, 1971.

103. **[Pearson1983a]** R.B. Pearson, J.L. Richardson, and D. Toussaint, "A Fast Processor for Monte Carlo Simulation," *J. Comput. Phys.*, vol. 51, p. 241, 1983.

104. **[Pearson1983b]** R.B. Pearson, "An Algorithm for Pseudo Random Number Generation Suitable for Large Scale Integration," *Journal of Computational Physics*, vol. 49, pp. 478-489, 1983.

105. **[Peterson1972]** W.W. Peterson and E.J. Weldon Jr., *Error-Correcting Codes,* The MIT Press, Cambridge, Mass., 1972.

106. **[Prange1957]** E. Prange, "Cyclic Error-Correcting Codes in Two Symbols," *AFCRC-TN-57*, vol. 103, Air Force Cambridge Research Center, Cambridge, Mass., Sept. 1957.

107. **[Pries1986]** W. Pries, A. Thanailakis, and H.C. Card, "Group Properties of Cellular Automata and VLSI Applications," *IEEE Trans. on Computers*, vol. C-35, pp. 1013-1024, Dec. 1986.

108. **[Pries1987]** W. Pries, *Personal Communication,* Dept. of Electrical Engineering, University of Manitoba, Winnipeg, MB, Canada, 1987.

109. **[Pries1988]** W. Pries, "Algebraic and Computational Aspects of Cellular Automata," *Ph.D. Dissertation*, University of Manitoba, Winnipeg, Canada, 1988.

110. **[Purdom1968]** P.W. Purdom and J.H. Williams, "Cycle Length in a Random Function," *Trans. of Amer. Math. Soc.*, vol. 133, No. 2, pp. 547-551, Sept. 1968.

111. **[Rapaport1985]** D.C. Rapaport, "Monte Carlo Experiments on Percolation: The Influences of Boundary Conditions," *Journal of Physics A*, vol. 18, p. L175, 1985.

112. **[Reddaway1985]** S.F. Reddaway, D.M. Scott, and K.A. Smith, "A Very High Speed Monte Carlo Simulation on DAP," *Computer Physics Communications*, vol. 37, p. 351, 1985.

113. **[Reddy1986]** M.K. Reddy and S.M. Reddy, "Detecting FET Stuck-Open Faults in CMOS Latches and Flip-Flops," *IEEE Design and Test*, vol. 3, No. 5, pp. 17-26, October 1986.

114. **[Reynolds1977]** P.J. Reynolds, H.E. Stanley, and W. Klein, *Journal of Phys. C*, vol. 10, pp. L167-L172, 1977.

115. **[Reynolds1978]** P.J. Reynolds, H.E. Stanley, and W. Klein, "Percolation by Position-Space Renormalisation Group with Large Cells," *Journal of Phys. A*, vol. 11, No. 8, pp. L199-L207, 1978.

116. **[Rodrigues1986]** A.B. Rodriguez-Vasquez, J.L. Huertas, and L.O. Chua, "Chaos in a Switched-Capacitor Circuit," *IEEE Trans. Circuits Syst.*, vol. CAS-32, pp. 1083-1085, Oct. 1986.

117. **[Rosen1987]** B.K. Rosen, *Personal Communication,* IBM T.J. Watson Research Center, Yorktown Heights, NewYork, 1987.

118. **[Roth1967]** J.P. Roth, W.G. Bouricius, and P.R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinquish Between Failures in Logic Circuits," *IEEE Transactions on Computers*, pp. 567-580, Oct. 1967.

119. **[Rubin1983]** S.R. Rubin, *An Integrated Aid for Top-Down Electrical Design,* Fairchild Laboratory for Artificial Intelligence Research, Palo Alto California, 1983.

120. **[Salem1986]** J. Salem and S. Wolfram , "Thermodynamics and Hydrodynamics of Cellular Automata," in *Theory and Applications of Cellular Automata*, ed. S. Wolfram, World Scientific, Singapore, 1986.

121. **[Savir1984]** J. Savir, G.S. Ditlow, and P.H. Bardell, "Random Pattern Testability," *IEEE Transactions on Computers*, pp. 79-90, Jan. 1984.

122. **[Schnurmann1975]** H.D. Schnurmann, E. Lindbloom, and R.G. Carpenter, "The Weighted Random Test-Pattern Generator," *IEEE Trans. on Computers*, vol. C-24, No. 7, pp. 695-700, July 1975.

123. **[Sedgewick1983]** R. Sedgewick, *Algorithms,* Addison Wesley, Reading Mass., 1983.

124. **[Seitz1984]** C. L. Seitz, "Concurrent VLSI Architectures," *IEEE Trans. on Computers*, vol. C-33, pp. 1247-1265, 1984.

125. **[Shante1971]** V.K. Shante and S. Kirkpatrick, "An Introduction to Percolation Theory," *Advaces in Physics*, vol. 20, pp. 325-357, 1971.

126. **[Smith1980]** J.E. Smith, "Measures of the Effectiveness of Fault Signature Analysis," *IEEE Trans. on Comp.*, vol. C-29, pp. 510-514, June 1980.

127. **[Smith1985]** K.A. Smith, S.F. Reddaway, and D.M. Scott, "Very High Performance Pseudorandom Number Generation on DAP," *Computer Physics Communications*, vol. 37, pp. 239-244, 1985.

128. **[Stanley1971]** H.E. Stanley, in *Introduction to Phase Transitions and Critical Phenomena*, Oxford University Press, New York, New York, 1971.

129. **[Stauffer1985]** D. Stauffer, *Introduction to Percolation Theory,* Taylor & Francis, Philadelphia, PA, 1985.

130. **[Stewart1977]** J.H. Stewart, "Future Testing of Large LSI Circuit Cards," *IEEE Semiconductor Test Symp.*, pp. 6-17, Oct. 1977.

131. **[Stoll1973]** E. Stoll, K. Binder, and T. Schneider, "Monte Carlo Investigation of Dynamic Critical Phenomena in the Two-Dimensional Kinetic Ising Model," *Phys. Rev. B*, vol. 8, No. 7, pp. 3266-3289, Oct. 1973.

132. **[Sur1976]** A. Sur, J.L. Lebowitz, J. Marro, M.H. Kalos, and S. Kirkpatrick, "Monte Carlo Studies of Percolation Phenomena for a Simple Cubic Lattice," *Journal of Stat. Phys.*, vol. 15, No. 5, pp. 345-353, 1976.

133. **[Swendsen1983]** R.H. Swendsen, "Monte Carlo Methods," in *Statistical and Particle Physics: Common Problems and Techniques*, ed. K.C. Bowler, A.J. McKane, Scottish Universities Summer School in Physics, Edinburgh, Scotland, 1983.

134. **[Tausworthe1965]** R.C. Tausworthe, "Random Numbers Generated by Linear Recurrance Modulo Two," *Mathematics of Computation*, vol. 19, pp. 201-209, 1965.

135. **[Thompson1980]** C.D. Thompson, "A Complexity Theory for VLSI," *Ph.D. Dissertation*, Carnegie-Mellon, 1980.

136. **[Thouless1978]** D.J. Thouless, "Percolation and Localization," in *Ill Condensed Matter*, ed. R. Balian, R. Maynard, G. Toulouse, pp. 1-62, World Scientific, Singapore, 1983.

137. **[Toffoli1984]** T. Toffoli, "CAM: A High Performance Cellular Automaton Machine," *Physica*, vol. 10D, p. 195, 1984.

138. **[Toffoli1987]** T. Toffoli and N. Margolus, *Cellular Automata Machines: A New Environment for Modeling,* The MIT Press, Cambridge, Mass., 1987.

139. **[TootIII1971]** J.P. Tootill, W.D. Robinson, and D.J. Eagle, "The Run up-and-down Performance of Tausworthe Pseudo-Random Number Generators," *Journal of the ACM*, vol. 18, No. 3, pp. 381-399, 1971.

140. **[TootIII1973]** J.P. Tootill, W.D. Robinson, A.G. Adams, and D.J. Eagle, "An Asymptotically Random Tausworthe Sequence," *Journal of the ACM*, vol. 18, No. 3, pp. 469-481, July 1973.

141. **[Tsui1987]** F.F. Tsui, *LSI/VLSI Testability Design,* McGraw-Hill, NewYork, NewYork, 1987.

142. **[Vecchi1983]** M.P. Vecchi and S. Kirkpatrick, "Global Wiring by Simulated Annealing," *IEEE Trans. on Computers*, vol. CAD-2, No. 4, pp. 215-222, Oct. 1983.

143. **[Verhangen1976]** A.M.W. Verhangen, "An Exactly Soluble Case of the Triangular Ising Model in a Magnetic Field," *J. Stat. Phys.*, vol. 15, p. 213, 1976.

144. **[Vichniac1984]** G. Vichniac, "Simulating Physics with Cellular Automata," *Physica*, vol. 10D, p. 96, 1984.

145. **[Wadsack1978]** R.L. Wadsack, "Fault Modeling and Logic Simulation of CMOS and MOS Integrated Circuits," *Bell System Technical Journal*, pp. 1449-1474, May-June 1978.

146. **[Wagner1987]** K.D. Wagner, C.K. Chin, and E.J. McCluskey, "Pseudorandom Testing," *IEEE Trans. on Computers*, vol. C-36, pp. 332-343, March 1987.

147. **[Waicukauski1987]** J.A. Waicukauski, E. Lindbloom, B.K. Rosen, and V.S. Iyengar, "Transition Fault Simulation," *IEEE Design and Test*, vol. 4, No. 2, pp. 32-38, April 1987.

148. **[Wallqvist1987]** A. Wallqvist, B.J. Berne, and C. Pangali, "Exploiting Physical Parallelism using Supercomputers: Two Examples from Chemical Physics," *IEEE Computer*, vol. 20, No. 5, pp. 9-21, May 1987.

149. **[Whittlesey1968]** J. Whittlesey, "A Comparison of the Correlation Behavior of Random Number Generators for the IBM 360," *Communications of the ACM*, vol.

11, No. 9, pp. 641-644, Sept. 1968.

150. **[Williams1983]** T.W. Williams and K.P. Parker, "Design for Testability - A Survey," *Proceedings of the IEEE*, vol. 71, pp. 98-112, 1983.

151. **[Williams1985]** T.W. Williams, "Test Length in a Self-Testing Environment," *IEEE Design and Test*, pp. 59-63, April 1985.

152. **[Williams1986a]** T.W. Williams, C.W. Starke, W. Daehn, and M. Gruetzner, "Comparison of Aliasing Probabilities for Primitive and Non-Primitive Polynomials," *Proc. of the 1986 IEEE International Test Conference*, pp. 282-288, 1986.

153. **[Williams1986b]** T.W. Williams, *VLSI Testing,* North Holland, NewYork, NewYork, 1986.

154. **[Wilson1975]** K.G. Wilson, "The Renormalization Group: Critical Phenomena and the Kondo Problem," *Rev. Mod. Phys.*, vol. 47, p. 765, 1975.

155. **[Wilson1979]** K.G. Wilson, "Problems in Physics with Many Scales of Length," *Sci. Amer.*, vol. 241, p. 158, 1979.

156. **[Wolfram1983]** S. Wolfram, "Statistical Mechanics of Cellular Automata," *Rev. of Modern Phys.*, vol. 55, pp. 601-644, 1983.

157. **[Wolfram1984a]** S. Wolfram, D. A. Lind, M. S. Waterman, P. Grassberger, and S. J. Willson, in *Cellular Automata, Proceedings of an Interdisciplinary Workshop*, North-Holland, Amsterdam, Los Alamos, N. M., 1984.

158. **[Wolfram1984b]** S. Wolfram, "Cellular Automata as Models for Complexity," *Nature*, vol. 311, p. 419, 1984.

159. **[Wolfram1984c]** S. Wolfram, "Universality and Complexity in Cellular Automata," *Physica*, vol. 10D, pp. 1-35, 1984.

160. **[Wolfram1984d]** S. Wolfram, "Computer Software in Science and Mathematics," *Sci. Amer.*, September 1984.

161. **[Wolfram1985a]** S. Wolfram, "Undecidability and Intractability in Theoretical Physics," *Phys. Rev. Lett.*, vol. 54, p. 735, 1985.

162. **[Wolfram1985b]** S. Wolfram, "Origins of Randomness in Physical Systems," *Phys. Rev. Lett.*, vol. 55, pp. 449-452, 1985.

163. **[Wolfram1986a]** S. Wolfram, "Random Sequence Generation by Cellular Automata," *Advances in Applied Mathematics*, vol. 7, pp. 127-169, 1986.

164. **[Zallen1983]** R. Zallen, *The Physics of Amorphous Solids,* John Wiley & Sons, New York, New York, 1983.

165. **[Zierler1969]** N. Zierler and J. Brillhart, "On Primitive Trinomials (mod 2) II," *Inform. Contr.*, vol. 14, No. 6, pp. 556-569, 1969.

Appendix A
Bad Sequence Probability

The following discussion concerns deriving the probability of a PRNG yielding a bad sequence given that a number of sequences have been found to be pseudorandom. This material is adapted from Papoulis [Papoulis1965].

Let the probability of an event Γ be p. If we conduct n trials then for any $\varepsilon > 0$ we have

$$\lim_{n\to\infty} P\left(\left| \frac{k}{n} - p \right| \le \varepsilon \right) = 1 \tag{A.1}$$

where k is the number of occurrences of Γ in n trials and $P(\ )$ is the probability of $\frac{k}{n}$ approaching p [Papoulis1965]. Consider a sequence of random variables $\langle x_i \rangle$ where

$$x_i = \begin{cases} 1 & \text{if } \Gamma \textit{ occurs in the } i\textit{'th trial} \\ 0 & \textit{otherwise} \end{cases} . \tag{A.2}$$

The mean of an infinite sequence $\langle x_i \rangle$ is

$$\begin{aligned} E(x) &= 1 \cdot P(x=1) + 0 \cdot P(x=0) \\ &= p \end{aligned} \tag{A.3}$$

and the variance is

$$\begin{aligned} \sigma^2 &= E(x^2) - E^2(x) \\ &= 1^2 \cdot P(x=1) + 0^2 \cdot P(x=0) - p^2 \\ &= p - p^2 \\ &= p(1-p) \ . \end{aligned} \tag{A.4}$$

If we consider the sequence to consist of only n trials then

$$\overline{x}_n = \frac{x_1 + x_2 + \cdots + x_n}{n} = \frac{k}{n} \tag{A.5}$$

where k and n are as defined above. Consider a collection of sequences $\langle x_i \rangle$. The expected value of $\overline{x}_n$ for this collection of sequences will be $E(\overline{x}_n) = p$. The Tchebycheff inequality states that if υ is arbitrary with density $f(\upsilon)$ and finite variance σ^2 then regardless of the shape of $f(\upsilon)$ the probability that υ lies in the interval $(\eta - \varepsilon, \eta + \varepsilon)$ is

$$P\left(\eta-\varepsilon < \upsilon < \eta+\varepsilon\right) \geq 1 - \frac{\sigma^2}{\varepsilon^2} \tag{A.6}$$

where $\eta = E(\upsilon)$. Therefore, using the values for $E(\overline{x}_n)$ and $\sigma^2_{\overline{x}_n}$ given above and Tchebycheff's inequality we can state that

$$P\left(\mid \overline{x}_n - p \mid < \varepsilon\right) \geq 1 - \frac{p(1-p)}{n\varepsilon^2}. \tag{A.7}$$

Since $p(1-p) \leq 1/4$ we finally conclude that

$$P\left(\mid \overline{x}_n - p \mid < \varepsilon\right) \geq 1 - \frac{1}{4n\varepsilon^2} \quad . \tag{A.8}$$

Thus, the sample mean, $\overline{x}_n$ tends to p. A much stronger statement due to Borel [Papoulis1965] states that the sample mean $\overline{x}_n$ tends to p with probability 1, i.e.

$$P(\overline{x}_n \rightarrow p) = 1 \quad \text{for } n \rightarrow \infty \quad . \tag{A.9}$$

Using this result we can see that if we test 100 sequences from a PRNG and find no bad sequences in the test and we want to estimate the probability, p, of the generator producing a bad sequence then for $\varepsilon = 0.1$ we have

$$P\left(\mid \overline{x}_n - p \mid \ < 0.1\right) \geq 1 - \frac{1}{400\times 0.1^2} = 0.75 \quad . \tag{A.10}$$

So we expect that in 75% of such tests the probability of a bad sequence occurring is less than 0.1. This of course gives a limited degree of confidence in the tested PRNG. If we wish to increase our confidence by reducing the probability estimate we must test many more sequences. For the **new** PRNGs suggested in this work such a test was made for 100,000 sequences from each PRNG yielding a 97.5% confidence that the probability of a bad sequence occurring is less than 0.01.

Appendix B
Complete Cycle Length Tables

The following tables give the complete number of cycles and percentage of states either in or on a path leading to each cycle. Cycle refers to the number of a particular cycle, size gives the cycle length, number indicates how many states are in the cycle or on paths leading to the cycle, and percent gives the percentage of states in the particular cycle.

B.1. Rule 30

length 4			
Cycle	size	number	percent
1	1	2	0.13
2	8	12	0.75
3	1	1	0.06
4	1	1	0.06

length 5			
Cycle	size	number	percent
1	1	2	0.06
2	5	30	0.94

length 6			
Cycle	size	number	percent
1	1	62	0.97
2	1	1	0.02
3	1	1	0.02

length 7			
Cycle	size	number	percent
1	1	2	0.02
2	4	7	0.06
3	4	7	0.06
4	63	77	0.60
5	4	7	0.06
6	4	7	0.06
7	4	7	0.06
8	4	7	0.06
9	4	7	0.06

length 8			
Cycle	size	number	percent
1	1	2	0.01
2	40	224	0.88
3	8	28	0.11
4	1	1	0.00
5	1	1	0.00

length 9			
Cycle	size	number	percent
1	1	8	0.02
2	72	90	0.18
3	171	414	0.81

length 10			
Cycle	size	number	percent
1	1	2	0.00
2	15	420	0.41
3	15	420	0.41
4	5	180	0.18
5	1	1	0.00
6	1	1	0.00

length 11			
Cycle	size	number	percent
1	1	2	0.00
2	154	1551	0.76
3	17	45	0.02
4	17	45	0.02
5	17	45	0.02
6	17	45	0.02
7	17	45	0.02
8	17	45	0.02
9	17	45	0.02
10	17	45	0.02
11	17	45	0.02
12	17	45	0.02
13	17	45	0.02

length 12			
Cycle	size	number	percent
1	1	242	0.06
2	102	957	0.23
3	102	957	0.23
4	102	957	0.23
5	102	957	0.23
6	3	3	0.00
7	8	12	0.00
8	3	3	0.00
9	3	3	0.00
10	1	1	0.00
11	3	3	0.00
12	1	1	0.00

length 13			
Cycle	size	number	percent
1	1	2	0.00
2	260	3575	0.44
3	832	2600	0.32
4	247	1924	0.23
5	91	91	0.01

length 14			
Cycle	size	number	percent
1	1	2	0.00
2	1428	13818	0.84
3	63	791	0.05
4	4	155	0.01
5	84	84	0.01
6	4	155	0.01
7	133	196	0.01
8	133	196	0.01
9	4	155	0.01
10	4	155	0.01
11	112	112	0.01
12	84	84	0.01
13	4	155	0.01
14	4	155	0.01
15	4	155	0.01
16	14	14	0.00
17	1	1	0.00
18	1	1	0.00

length 15			
Cycle	size	number	percent
1	1	8	0.00
2	1455	30375	0.93
3	9	276	0.01
4	9	276	0.01
5	9	276	0.01
6	9	276	0.01
7	30	171	0.01
8	30	171	0.01
9	9	276	0.01
10	30	171	0.01
11	30	171	0.01
12	7	7	0.00
13	30	171	0.01
14	7	7	0.00
15	7	7	0.00
16	5	30	0.00
17	7	7	0.00
18	7	7	0.00
19	7	7	0.00
20	7	7	0.00
21	7	7	0.00
22	5	5	0.00
23	7	7	0.00
24	7	7	0.00
25	7	7	0.00
26	5	5	0.00
27	7	7	0.00
28	7	7	0.00
29	5	5	0.00
30	7	7	0.00
31	7	7	0.00

Complete Cycle Length Tables

length 16			
Cycle	size	number	percent
1	1	2	0.00
2	6016	32496	0.50
3	40	4432	0.07
4	4144	27056	0.41
5	8	1468	0.02
6	40	40	0.00
7	40	40	0.00
8	1	1	0.00
9	1	1	0.00

length 17			
Cycle	size	number	percent
1	1	2	0.00
2	10846	125375	0.96
3	1632	3434	0.03
4	867	1802	0.01
5	306	306	0.00
6	136	136	0.00
7	17	17	0.00

length 18			
Cycle	size	number	percent
1	1	62	0.00
2	2844	213930	0.82
3	171	37422	0.14
4	186	1320	0.01
5	186	1320	0.01
6	186	1320	0.01
7	186	1320	0.01
8	186	1320	0.01
9	186	1320	0.01
10	72	2664	0.01
11	24	24	0.00
12	24	24	0.00
13	24	24	0.00
14	24	24	0.00
15	24	24	0.00
16	24	24	0.00
17	1	1	0.00
18	1	1	0.00

length 19			
Cycle	size	number	percent
1	1	2	0.00
2	247	378233	0.72
3	3705	145844	0.28
4	133	171	0.00
5	38	38	0.00

length 20			
Cycle	size	number	percent
1	1	2	0.00
2	3420	32560	0.03
3	3420	32560	0.03
4	6150	278237	0.27
5	3420	32560	0.03
6	6150	308674	0.29
7	3420	32560	0.03
8	15	68330	0.07
9	5	104500	0.10
10	15	68330	0.07
11	1715	8645	0.01
12	1715	8645	0.01
13	1715	8645	0.01
14	1715	8645	0.01
15	68	456	0.00
16	6691	41483	0.04
17	68	456	0.00
18	68	456	0.00
19	580	740	0.00
20	68	456	0.00
21	68	456	0.00
22	6756	11046	0.01
23	30	30	0.00
24	30	30	0.00
25	8	12	0.00
26	30	30	0.00
27	30	30	0.00
28	1	1	0.00
29	1	1	0.00

B.2. Rule 45

length 4			
Cycle	size	number	percent
1	2	16	1.00

length 5			
Cycle	size	number	percent
1	2	2	0.06
2	30	30	0.94

length 6			
Cycle	size	number	percent
1	2	4	0.06
2	18	54	0.84
3	1	1	0.02
4	1	1	0.02
5	3	3	0.05
6	1	1	0.02

length 7			
Cycle	size	number	percent
1	2	2	0.02
2	126	126	0.98

length 8			
Cycle	size	number	percent
1	2	152	0.59
2	32	32	0.13
3	24	24	0.09
4	16	16	0.06
5	24	24	0.09
6	4	4	0.02
7	4	4	0.02

length 9			
Cycle	size	number	percent
1	2	2	0.00
2	504	504	0.98
3	1	1	0.00
4	1	1	0.00
5	3	3	0.01
6	1	1	0.00

length 10			
Cycle	size	number	percent
1	2	4	0.00
2	430	720	0.70
3	60	60	0.06
4	60	60	0.06
5	30	30	0.03
6	15	15	0.02
7	60	60	0.06
8	60	60	0.06
9	15	15	0.02

length 11			
Cycle	size	number	percent
1	2	2	0.00
2	979	979	0.48
3	935	935	0.46
4	5	5	0.00
5	5	5	0.00
6	66	66	0.03
7	5	5	0.00
8	5	5	0.00
9	5	5	0.00
10	11	11	0.01
11	5	5	0.00
12	5	5	0.00
13	5	5	0.00
14	5	5	0.00
15	5	5	0.00
16	5	5	0.00

Complete Cycle Length Tables

length 12			
Cycle	size	number	percent
1	2	16	0.00
2	18	2706	0.66
3	156	612	0.15
4	24	24	0.01
5	240	240	0.06
6	24	24	0.01
7	24	24	0.01
8	24	24	0.01
9	12	12	0.00
10	24	24	0.01
11	24	24	0.01
12	12	12	0.00
13	84	84	0.02
14	12	12	0.00
15	12	12	0.00
16	24	24	0.01
17	24	24	0.01
18	12	12	0.00
19	24	24	0.01
20	24	24	0.01
21	12	12	0.00
22	12	12	0.00
23	24	24	0.01
24	12	12	0.00
25	12	12	0.00
26	1	1	0.00
27	12	12	0.00
28	12	12	0.00
29	24	24	0.01
30	12	12	0.00
31	1	1	0.00
32	3	3	0.00
33	1	1	0.00

length 13			
Cycle	size	number	percent
1	2	2	0.00
2	676	676	0.08
3	443	443	0.05
4	443	443	0.05
5	443	443	0.05
6	443	443	0.05
7	443	443	0.05
8	1105	1105	0.13
9	443	443	0.05
10	443	443	0.05
11	443	443	0.05
12	443	443	0.05
13	443	443	0.05
14	443	443	0.05
15	443	443	0.05
16	443	443	0.05
17	39	39	0.00
18	130	130	0.02
19	78	78	0.01
20	78	78	0.01
21	156	156	0.02
22	78	78	0.01
23	78	78	0.01
24	13	13	0.00

length 14			
Cycle	size	number	percent
1	2	4	0.00
2	2198	8568	0.52
3	534	628	0.04
4	534	628	0.04
5	534	628	0.04
6	534	628	0.04
7	534	628	0.04
8	534	628	0.04
9	392	1435	0.09
10	534	628	0.04
11	392	1435	0.09
12	168	168	0.01
13	126	126	0.01
14	168	168	0.01
15	42	42	0.00
16	42	42	0.00

length 15			
Cycle	size	number	percent
1	2	2	0.00
2	2340	2340	0.07
3	6820	6820	0.21
4	32	32	0.00
5	6820	6820	0.21
6	6820	6820	0.21
7	32	32	0.00
8	4920	4920	0.15
9	32	32	0.00
10	32	32	0.00
11	2820	2820	0.09
12	32	32	0.00
13	32	32	0.00
14	120	120	0.00
15	32	32	0.00
16	32	32	0.00
17	120	120	0.00
18	32	32	0.00
19	32	32	0.00
20	120	120	0.00
21	60	60	0.00
22	32	32	0.00
23	32	32	0.00
24	30	30	0.00
25	30	30	0.00
26	60	60	0.00
27	60	60	0.00
28	30	30	0.00
29	30	30	0.00
30	60	60	0.00
31	60	60	0.00
32	60	60	0.00
33	60	60	0.00
34	60	60	0.00
35	60	60	0.00
36	60	60	0.00
37	60	60	0.00
38	60	60	0.00
39	32	32	0.00
40	60	60	0.00
41	60	60	0.00
42	60	60	0.00
43	60	60	0.00
44	60	60	0.00
45	60	60	0.00
46	60	60	0.00
47	60	60	0.00
48	60	60	0.00
49	32	32	0.00
50	32	32	0.00
51	1	1	0.00
52	1	1	0.00
53	3	3	0.00
54	1	1	0.00

length 16			
Cycle	size	number	percent
1	2	39448	0.60
2	976	12304	0.19
3	700	1100	0.02
4	556	556	0.01
5	848	848	0.01
6	700	1100	0.02
7	700	1100	0.02
8	556	556	0.01
9	2816	3712	0.06
10	700	1100	0.02
11	556	556	0.01
12	296	296	0.00
13	556	556	0.01
14	296	296	0.00
15	296	296	0.00
16	208	208	0.00
17	296	296	0.00
18	144	144	0.00
19	144	144	0.00
20	32	32	0.00
21	48	48	0.00
22	48	48	0.00
23	24	24	0.00
24	48	48	0.00
25	48	48	0.00
26	48	48	0.00
27	48	48	0.00
28	48	48	0.00
29	48	48	0.00
30	48	48	0.00
31	48	48	0.00
32	48	48	0.00
33	48	48	0.00
34	48	48	0.00
35	16	16	0.00
36	48	48	0.00
37	24	24	0.00
38	48	48	0.00
39	48	48	0.00
40	48	48	0.00
41	4	4	0.00
42	4	4	0.00

length 17			
Cycle	size	number	percent
1	2	2	0.00
2	78812	78812	0.60
3	2176	2176	0.02
4	6052	6052	0.05
5	32912	32912	0.25
6	4845	4845	0.04
7	867	867	0.01
8	408	408	0.00
9	408	408	0.00
10	408	408	0.00
11	408	408	0.00
12	408	408	0.00
13	816	816	0.01
14	408	408	0.00
15	204	204	0.00
16	204	204	0.00
17	204	204	0.00
18	204	204	0.00
19	408	408	0.00
20	408	408	0.00
21	408	408	0.00
22	102	102	0.00

length 18			
Cycle	size	number	percent
1	2	0	0.00
2	3756	37008	0.14
3	3756	37008	0.14
4	8787	48294	0.18
5	3756	37008	0.14
6	7812	48294	0.18
7	7812	35068	0.13
8	8168	13226	0.05
9	504	504	0.00
10	72	72	0.00
11	36	36	0.00
12	72	72	0.00
13	36	36	0.00
14	72	72	0.00
15	36	36	0.00
16	72	72	0.00
17	36	36	0.00
18	36	36	0.00
19	18	54	0.00
20	72	72	0.00
21	36	36	0.00
22	72	72	0.00
23	21	21	0.00
24	36	36	0.00
25	72	72	0.00
26	36	36	0.00
27	72	72	0.00
28	72	72	0.00
29	36	36	0.00
30	21	21	0.00
31	18	18	0.00
32	18	18	0.00
33	18	18	0.00
34	18	18	0.00
35	36	36	0.00
36	18	18	0.00
37	18	18	0.00
38	18	18	0.00
39	18	18	0.00
40	36	36	0.00
41	36	36	0.00
42	18	18	0.00
43	18	18	0.00
44	18	18	0.00
45	18	18	0.00
46	36	36	0.00
47	18	18	0.00
48	18	18	0.00
49	18	18	0.00
50	18	18	0.00
51	36	36	0.00
52	18	18	0.00
53	18	18	0.00
54	36	36	0.00
55	36	36	0.00
56	18	18	0.00
57	18	18	0.00
58	36	36	0.00
59	36	36	0.00
60	72	72	0.00
61	36	36	0.00
62	36	36	0.00
63	36	36	0.00

64	36	36	0.00
65	36	36	0.00
66	36	36	0.00
67	36	36	0.00
68	18	18	0.00
69	18	18	0.00
70	36	36	0.00
71	18	18	0.00
72	18	18	0.00
73	36	36	0.00
74	18	18	0.00
75	18	18	0.00
76	36	36	0.00
77	18	18	0.00
78	18	18	0.00
79	36	36	0.00
80	36	36	0.00
81	18	18	0.00
82	18	18	0.00
83	36	36	0.00
84	18	18	0.00
85	18	18	0.00
86	18	18	0.00
87	18	18	0.00
88	36	36	0.00
89	21	21	0.00
90	18	18	0.00
91	18	18	0.00
92	36	36	0.00
93	18	18	0.00
94	18	18	0.00
95	18	18	0.00
96	18	18	0.00
97	36	36	0.00
98	36	36	0.00
99	36	36	0.00
100	36	36	0.00
101	36	36	0.00
102	36	36	0.00
103	36	36	0.00
104	36	36	0.00
105	36	36	0.00
106	36	36	0.00
107	36	36	0.00
108	36	36	0.00
109	36	36	0.00
110	36	36	0.00
111	36	36	0.00
112	72	72	0.00
113	36	36	0.00
114	18	18	0.00
115	18	18	0.00
116	18	18	0.00
117	18	18	0.00
118	36	36	0.00
119	36	36	0.00
120	18	18	0.00
121	18	18	0.00
122	18	18	0.00
123	18	18	0.00
124	18	18	0.00
125	18	18	0.00
126	36	36	0.00
127	72	72	0.00
128	36	36	0.00
129	36	36	0.00
130	18	18	0.00
131	18	18	0.00
132	36	36	0.00
133	18	18	0.00
134	18	18	0.00
135	18	18	0.00
136	18	18	0.00
137	18	18	0.00
138	36	36	0.00
139	21	21	0.00
140	18	18	0.00
141	18	18	0.00
142	18	18	0.00
143	36	36	0.00
144	36	36	0.00
145	36	36	0.00
146	36	36	0.00
147	36	36	0.00
148	36	36	0.00
149	36	36	0.00
150	18	18	0.00
151	18	18	0.00
152	18	18	0.00
153	18	18	0.00
154	18	18	0.00
155	18	18	0.00
156	18	18	0.00
157	18	18	0.00
158	36	36	0.00
159	18	18	0.00
160	18	18	0.00
161	36	36	0.00
162	36	36	0.00
163	36	36	0.00
164	18	18	0.00
165	18	18	0.00
166	18	18	0.00
167	18	18	0.00
168	36	36	0.00
169	36	36	0.00
170	18	18	0.00
171	18	18	0.00
172	18	18	0.00
173	18	18	0.00
174	18	18	0.00
175	18	18	0.00
176	21	21	0.00
177	36	36	0.00
178	36	36	0.00
179	36	36	0.00
180	36	36	0.00
181	36	36	0.00
182	36	36	0.00
183	36	36	0.00
184	36	36	0.00
185	36	36	0.00
186	36	36	0.00
187	36	36	0.00
188	36	36	0.00
189	36	36	0.00
190	36	36	0.00
191	21	21	0.00
192	1	1	0.00
193	36	36	0.00

194	36	36	0.00
195	36	36	0.00
196	36	36	0.00
197	36	36	0.00
198	36	36	0.00
199	36	36	0.00
200	1	1	0.00
201	2	4	0.00
202	3	3	0.00
203	1	1	0.00

length 19			
Cycle	size	number	percent
1	2	2	0.00
2	183920	183920	0.35
3	158080	158080	0.30
4	149435	149435	0.29
5	15371	15371	0.03
6	3458	3458	0.01
7	1425	1425	0.00
8	1653	1653	0.00
9	456	456	0.00
10	361	361	0.00
11	912	912	0.00
12	912	912	0.00
13	912	912	0.00
14	912	912	0.00
15	456	456	0.00
16	456	456	0.00
17	456	456	0.00
18	456	456	0.00
19	456	456	0.00
20	456	456	0.00
21	912	912	0.00
22	95	95	0.00
23	456	456	0.00
24	456	456	0.00
25	456	456	0.00
26	114	114	0.00
27	114	114	0.00
28	114	114	0.00
29	114	114	0.00
30	228	228	0.00
31	114	114	0.00
32	114	114	0.00
33	114	114	0.00
34	114	114	0.00
35	114	114	0.00
36	114	114	0.00

length 20			
Cycle	size	number	percent
1	2	16	0.00
2	142580	501000	0.48
3	14265	65870	0.06
4	430	208240	0.20
5	9112	9112	0.01
6	14265	65870	0.06
7	9112	9112	0.01
8	14265	65870	0.06
9	14265	65870	0.06
10	9112	9112	0.01
11	9112	9112	0.01
12	9112	9112	0.01
13	252	252	0.00
14	252	252	0.00
15	252	252	0.00
16	236	236	0.00
17	236	236	0.00
18	120	120	0.00
19	4260	4260	0.00
20	236	236	0.00
21	1100	1100	0.00
22	252	252	0.00
23	236	236	0.00
24	120	120	0.00
25	236	236	0.00
26	252	252	0.00
27	280	280	0.00
28	120	120	0.00
29	480	480	0.00
30	240	240	0.00
31	120	120	0.00
32	120	120	0.00
33	120	120	0.00
34	120	120	0.00
35	120	120	0.00
36	120	120	0.00
37	120	120	0.00
38	120	120	0.00
39	120	120	0.00
40	120	120	0.00
41	240	240	0.00
42	120	120	0.00
43	240	240	0.00
44	120	120	0.00
45	120	120	0.00
46	60	60	0.00
47	280	280	0.00
48	120	120	0.00
49	280	280	0.00
50	120	120	0.00
51	120	120	0.00
52	120	120	0.00
53	120	120	0.00
54	120	120	0.00
55	120	120	0.00
56	120	120	0.00
57	120	120	0.00
58	120	120	0.00
59	120	120	0.00
60	120	120	0.00
61	240	240	0.00
62	120	120	0.00
63	240	240	0.00

64	280	280	0.00
65	120	120	0.00
66	120	120	0.00
67	120	120	0.00
68	60	60	0.00
69	120	120	0.00
70	120	120	0.00
71	120	120	0.00
72	120	120	0.00
73	120	120	0.00
74	120	120	0.00
75	120	120	0.00
76	120	120	0.00
77	120	120	0.00
78	120	120	0.00
79	240	240	0.00
80	120	120	0.00
81	120	120	0.00
82	60	60	0.00
83	60	60	0.00
84	60	60	0.00
85	60	60	0.00
86	120	120	0.00
87	60	60	0.00
88	60	60	0.00
89	60	60	0.00
90	240	240	0.00
91	60	60	0.00
92	60	60	0.00
93	60	60	0.00
94	120	120	0.00
95	120	120	0.00
96	120	120	0.00
97	30	30	0.00
98	60	60	0.00
99	60	60	0.00
100	60	60	0.00
101	60	60	0.00
102	60	60	0.00
103	60	60	0.00
104	60	60	0.00
105	60	60	0.00
106	60	60	0.00
107	60	60	0.00
108	60	60	0.00
109	60	60	0.00
110	60	60	0.00
111	60	60	0.00
112	60	60	0.00
113	60	60	0.00
114	60	60	0.00
115	60	60	0.00
116	60	60	0.00
117	60	60	0.00
118	60	60	0.00
119	60	60	0.00
120	60	60	0.00
121	60	60	0.00
122	60	60	0.00
123	60	60	0.00
124	60	60	0.00
125	60	60	0.00
126	60	60	0.00
127	60	60	0.00
128	60	60	0.00
129	60	60	0.00
130	15	15	0.00
131	60	60	0.00
132	60	60	0.00
133	60	60	0.00
134	60	60	0.00
135	60	60	0.00
136	60	60	0.00
137	60	60	0.00
138	60	60	0.00
139	60	60	0.00
140	60	60	0.00
141	60	60	0.00
142	60	60	0.00
143	60	60	0.00
144	60	60	0.00
145	60	60	0.00
146	60	60	0.00
147	60	60	0.00
148	60	60	0.00
149	60	60	0.00
150	60	60	0.00
151	60	60	0.00
152	60	60	0.00
153	60	60	0.00
154	60	60	0.00
155	120	120	0.00
156	60	60	0.00
157	120	120	0.00
158	60	60	0.00
159	120	120	0.00
160	240	240	0.00
161	120	120	0.00
162	60	60	0.00
163	120	120	0.00
164	60	60	0.00
165	120	120	0.00
166	60	60	0.00
167	120	120	0.00
168	120	120	0.00
169	60	60	0.00
170	120	120	0.00
171	120	120	0.00
172	120	120	0.00
173	120	120	0.00
174	120	120	0.00
175	120	120	0.00
176	120	120	0.00
177	120	120	0.00
178	120	120	0.00
179	240	240	0.00
180	60	60	0.00
181	60	60	0.00
182	60	60	0.00
183	120	120	0.00
184	60	60	0.00
185	60	60	0.00
186	60	60	0.00
187	60	60	0.00
188	60	60	0.00
189	60	60	0.00
190	60	60	0.00
191	60	60	0.00
192	60	60	0.00
193	60	60	0.00

194	60	60	0.00
195	60	60	0.00
196	60	60	0.00
197	60	60	0.00
198	60	60	0.00
199	60	60	0.00
200	60	60	0.00
201	60	60	0.00
202	60	60	0.00
203	60	60	0.00
204	60	60	0.00
205	60	60	0.00
206	60	60	0.00
207	60	60	0.00
208	60	60	0.00
209	60	60	0.00
210	60	60	0.00
211	60	60	0.00
212	60	60	0.00
213	60	60	0.00
214	120	120	0.00
215	30	30	0.00
216	60	60	0.00
217	60	60	0.00
218	60	60	0.00
219	60	60	0.00
220	60	60	0.00
221	60	60	0.00
222	60	60	0.00
223	60	60	0.00
224	60	60	0.00
225	30	30	0.00
226	60	60	0.00
227	60	60	0.00
228	120	120	0.00
229	60	60	0.00
230	60	60	0.00
231	60	60	0.00
232	60	60	0.00
233	60	60	0.00
234	60	60	0.00
235	60	60	0.00
236	60	60	0.00
237	60	60	0.00
238	60	60	0.00
239	30	30	0.00
240	60	60	0.00
241	60	60	0.00
242	60	60	0.00
243	60	60	0.00
244	60	60	0.00
245	60	60	0.00
246	60	60	0.00
247	60	60	0.00
248	60	60	0.00
249	60	60	0.00
250	120	120	0.00
251	60	60	0.00
252	120	120	0.00
253	60	60	0.00
254	60	60	0.00
255	60	60	0.00
256	60	60	0.00
257	60	60	0.00
258	60	60	0.00
259	60	60	0.00
260	60	60	0.00
261	60	60	0.00
262	60	60	0.00
263	120	120	0.00
264	15	15	0.00
265	60	60	0.00
266	60	60	0.00
267	60	60	0.00
268	60	60	0.00
269	60	60	0.00
270	60	60	0.00
271	60	60	0.00
272	60	60	0.00
273	60	60	0.00
274	120	120	0.00
275	60	60	0.00
276	60	60	0.00
277	60	60	0.00
278	60	60	0.00
279	60	60	0.00
280	60	60	0.00
281	30	30	0.00
282	60	60	0.00

B.3. Rule 30 and 45 Hybrid

length 4			
Cycle	size	number	percent
1	7	15	0.94
2	1	1	0.06

length 5			
Cycle	size	number	percent
1	4	32	1.00

length 6			
Cycle	size	number	percent
1	14	64	1.00

length 7			
Cycle	size	number	percent
1	13	120	0.94
2	6	7	0.06
3	1	1	0.01

length 8			
Cycle	size	number	percent
1	30	160	0.63
2	35	87	0.34
3	8	8	0.03
4	1	1	0.00

length 9			
Cycle	size	number	percent
1	15	512	1.00

length 10			
Cycle	size	number	percent
1	335	950	0.93
2	45	45	0.04
3	13	13	0.01
4	16	16	0.02

length 11			
Cycle	size	number	percent
1	27	1994	0.97
2	22	39	0.02
3	14	14	0.01
4	1	1	0.00

length 12			
Cycle	size	number	percent
1	311	2184	0.53
2	111	178	0.04
3	12	289	0.07
4	101	1276	0.31
5	5	144	0.04
6	12	12	0.00
7	12	12	0.00
8	1	1	0.00

length 13			
Cycle	size	number	percent
1	263	5831	0.71
2	231	1287	0.16
3	281	1059	0.13
4	15	15	0.00

length 14			
Cycle	size	number	percent
1	543	11150	0.68
2	61	4938	0.30
3	100	243	0.02
4	32	32	0.00
5	16	16	0.00
6	5	5	0.00

length 15			
Cycle	size	number	percent
1	993	7604	0.23
2	1211	25148	0.77
3	15	15	0.00
4	1	1	0.00

length 16			
Cycle	size	number	percent
1	1090	19887	0.30
2	4962	43767	0.67
3	1060	1604	0.02
4	196	259	0.00
5	10	10	0.00
6	1	1	0.00
7	8	8	0.00

length 17			
Cycle	size	number	percent
1	6183	73794	0.56
2	1147	57151	0.44
3	98	98	0.00
4	10	10	0.00
5	19	19	0.00

length 18			
Cycle	size	number	percent
1	1318	172564	0.66
2	4174	60669	0.23
3	4454	21683	0.08
4	644	7056	0.03
5	90	90	0.00
6	56	56	0.00
7	5	5	0.00
8	14	14	0.00
9	7	7	0.00

length 19			
Cycle	size	number	percent
1	544	55691	0.11
2	140	4281	0.01
3	4795	157792	0.30
4	1755	151494	0.29
5	2156	111056	0.21
6	4834	38189	0.07
7	2042	5125	0.01
8	235	556	0.00
9	18	18	0.00
10	50	50	0.00
11	7	7	0.00
12	22	22	0.00
13	6	6	0.00
14	1	1	0.00

length 20			
Cycle	size	number	percent
1	1246	390766	0.37
2	1331	590328	0.56
3	11413	33887	0.03
4	3309	30463	0.03
5	1723	1723	0.00
6	48	50	0.00
7	270	1296	0.00
8	57	57	0.00
9	1	1	0.00
10	5	5	0.00

Appendix C
Bit Weight Tables

The following tables give a complete listing of the weight of each output bit for all possible simple one-dimensional CAs. Here the weight of high bits as a function of position in the CA is given as a number pair xxx (yy), where xxx refers to the fraction of bits which are high out of the total number of bits occuring in CA position yy.

rule 1
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.000 (14)
0.500 (15) 0.500 (16) 0.000 (17) 0.500 (18) 0.500 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.500 (28) 0.500 (29)
Average = 0.467 Range = 0.500

rule 2
0.067 (0) 0.067 (1) 0.067 (2) 0.067 (3) 0.067 (4)
0.067 (5) 0.067 (6) 0.067 (7) 0.067 (8) 0.067 (9)
0.067 (10) 0.067 (11) 0.067 (12) 0.067 (13) 0.067 (14)
0.067 (15) 0.067 (16) 0.067 (17) 0.067 (18) 0.067 (19)
0.067 (20) 0.067 (21) 0.067 (22) 0.067 (23) 0.067 (24)
0.067 (25) 0.067 (26) 0.067 (27) 0.067 (28) 0.067 (29)
Average = 0.067 Range = 0.000

rule 3
0.433 (0) 0.433 (1) 0.433 (2) 0.433 (3) 0.433 (4)
0.433 (5) 0.433 (6) 0.433 (7) 0.433 (8) 0.433 (9)
0.433 (10) 0.433 (11) 0.433 (12) 0.433 (13) 0.433 (14)
0.433 (15) 0.433 (16) 0.433 (17) 0.433 (18) 0.433 (19)
0.433 (20) 0.433 (21) 0.433 (22) 0.433 (23) 0.433 (24)
0.433 (25) 0.433 (26) 0.433 (27) 0.433 (28) 0.433 (29)
Average = 0.433 Range = 0.000

rule 4
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 1.000 (4)
0.000 (5) 0.000 (6) 1.000 (7) 0.000 (8) 1.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
1.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
0.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 1.000 (29)
Average = 0.167 Range = 1.000

rule 5
0.000 (0) 0.500 (1) 0.500 (2) 0.000 (3) 1.000 (4)
0.000 (5) 0.000 (6) 1.000 (7) 0.000 (8) 1.000 (9)
0.000 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.000 (14)
1.000 (15) 0.000 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.000 (26) 1.000 (27) 0.000 (28) 1.000 (29)
Average = 0.433 Range = 1.000

rule 6
0.200 (0) 0.300 (1) 0.200 (2) 0.300 (3) 0.200 (4)
0.300 (5) 0.200 (6) 0.300 (7) 0.200 (8) 0.300 (9)
0.200 (10) 0.300 (11) 0.200 (12) 0.300 (13) 0.200 (14)
0.300 (15) 0.200 (16) 0.300 (17) 0.200 (18) 0.300 (19)
0.200 (20) 0.300 (21) 0.200 (22) 0.300 (23) 0.200 (24)
0.300 (25) 0.200 (26) 0.300 (27) 0.200 (28) 0.300 (29)
Average = 0.250 Range = 0.100

rule 7
0.450 (0) 0.450 (1) 0.450 (2) 0.450 (3) 0.450 (4)
0.450 (5) 0.450 (6) 0.450 (7) 0.450 (8) 0.450 (9)
0.450 (10) 0.450 (11) 0.450 (12) 0.450 (13) 0.450 (14)
0.450 (15) 0.450 (16) 0.450 (17) 0.450 (18) 0.450 (19)
0.450 (20) 0.450 (21) 0.450 (22) 0.450 (23) 0.450 (24)
0.450 (25) 0.450 (26) 0.450 (27) 0.450 (28) 0.450 (29)
Average = 0.450 Range = 0.001

rule 8
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 0.000 (4)
0.000 (5) 0.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
0.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 0.000 (29)
Average = 0.000 Range = 0.000

rule 9
0.433 (0) 0.433 (1) 0.433 (2) 0.433 (3) 0.433 (4)
0.433 (5) 0.433 (6) 0.433 (7) 0.433 (8) 0.433 (9)
0.433 (10) 0.433 (11) 0.433 (12) 0.433 (13) 0.433 (14)
0.433 (15) 0.433 (16) 0.433 (17) 0.433 (18) 0.433 (19)
0.433 (20) 0.433 (21) 0.433 (22) 0.433 (23) 0.433 (24)
0.433 (25) 0.433 (26) 0.433 (27) 0.433 (28) 0.433 (29)
Average = 0.433 Range = 0.001

rule 10
0.300 (0) 0.300 (1) 0.300 (2) 0.300 (3) 0.300 (4)
0.300 (5) 0.300 (6) 0.300 (7) 0.300 (8) 0.300 (9)
0.300 (10) 0.300 (11) 0.300 (12) 0.300 (13) 0.300 (14)
0.300 (15) 0.300 (16) 0.300 (17) 0.300 (18) 0.300 (19)
0.300 (20) 0.300 (21) 0.300 (22) 0.300 (23) 0.300 (24)
0.300 (25) 0.300 (26) 0.300 (27) 0.300 (28) 0.300 (29)
Average = 0.300 Range = 0.000

rule 11
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.500 (28) 0.500 (29)
Average = 0.500 Range = 0.000

rule 12
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 1.000 (4)
0.000 (5) 0.000 (6) 1.000 (7) 0.000 (8) 1.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
1.000 (15) 0.000 (16) 1.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 1.000 (24)
0.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 1.000 (29)
Average = 0.233 Range = 1.000

rule 13
0.000 (0) 0.000 (1) 1.000 (2) 0.000 (3) 1.000 (4)
0.000 (5) 0.000 (6) 1.000 (7) 0.000 (8) 0.000 (9)
1.000 (10) 0.000 (11) 1.000 (12) 0.000 (13) 1.000 (14)
0.000 (15) 1.000 (16) 0.000 (17) 1.000 (18) 0.000 (19)
0.000 (20) 1.000 (21) 0.000 (22) 0.000 (23) 1.000 (24)
0.000 (25) 1.000 (26) 0.000 (27) 0.000 (28) 1.000 (29)
Average = 0.400 Range = 1.000

rule 14
0.467 (0) 0.533 (1) 0.467 (2) 0.533 (3) 0.467 (4)
0.533 (5) 0.467 (6) 0.533 (7) 0.467 (8) 0.533 (9)
0.467 (10) 0.533 (11) 0.467 (12) 0.533 (13) 0.467 (14)
0.533 (15) 0.467 (16) 0.533 (17) 0.467 (18) 0.533 (19)
0.467 (20) 0.533 (21) 0.467 (22) 0.533 (23) 0.467 (24)
0.533 (25) 0.467 (26) 0.534 (27) 0.467 (28) 0.533 (29)
Average = 0.500 Range = 0.067

rule 15
0.600 (0) 0.400 (1) 0.600 (2) 0.400 (3) 0.600 (4)
0.400 (5) 0.600 (6) 0.400 (7) 0.600 (8) 0.400 (9)
0.600 (10) 0.400 (11) 0.600 (12) 0.400 (13) 0.600 (14)
0.400 (15) 0.600 (16) 0.400 (17) 0.600 (18) 0.400 (19)
0.600 (20) 0.400 (21) 0.600 (22) 0.400 (23) 0.600 (24)
0.400 (25) 0.600 (26) 0.400 (27) 0.600 (28) 0.400 (29)
Average = 0.500 Range = 0.200

rule 16
0.100 (0) 0.100 (1) 0.100 (2) 0.100 (3) 0.100 (4)
0.100 (5) 0.100 (6) 0.100 (7) 0.100 (8) 0.100 (9)
0.100 (10) 0.100 (11) 0.100 (12) 0.100 (13) 0.100 (14)
0.100 (15) 0.100 (16) 0.100 (17) 0.100 (18) 0.100 (19)
0.100 (20) 0.100 (21) 0.100 (22) 0.100 (23) 0.100 (24)
0.100 (25) 0.100 (26) 0.100 (27) 0.100 (28) 0.100 (29)
Average = 0.100 Range = 0.000

rule 17
0.417 (0) 0.417 (1) 0.417 (2) 0.417 (3) 0.417 (4)
0.417 (5) 0.417 (6) 0.417 (7) 0.417 (8) 0.417 (9)
0.417 (10) 0.417 (11) 0.417 (12) 0.417 (13) 0.417 (14)
0.417 (15) 0.417 (16) 0.417 (17) 0.417 (18) 0.417 (19)
0.417 (20) 0.417 (21) 0.417 (22) 0.417 (23) 0.417 (24)
0.417 (25) 0.417 (26) 0.417 (27) 0.417 (28) 0.417 (29)
Average = 0.417 Range = 0.000

rule 18
0.001 (0) 0.001 (1) 0.001 (2) 0.001 (3) 0.001 (4)
0.001 (5) 0.001 (6) 0.001 (7) 0.001 (8) 0.001 (9)
0.001 (10) 0.000 (11) 0.001 (12) 0.001 (13) 0.000 (14)
0.001 (15) 0.001 (16) 0.000 (17) 0.001 (18) 0.001 (19)
0.001 (20) 0.001 (21) 0.001 (22) 0.001 (23) 0.001 (24)
0.001 (25) 0.001 (26) 0.000 (27) 0.001 (28) 0.001 (29)
Average = 0.001 Range = 0.001

rule 19
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.500 (28) 0.500 (29)
Average = 0.500 Range = 0.000

rule 20
0.267 (0) 0.333 (1) 0.267 (2) 0.333 (3) 0.267 (4)
0.333 (5) 0.267 (6) 0.333 (7) 0.267 (8) 0.333 (9)
0.267 (10) 0.333 (11) 0.267 (12) 0.333 (13) 0.267 (14)
0.333 (15) 0.267 (16) 0.333 (17) 0.267 (18) 0.333 (19)
0.266 (20) 0.333 (21) 0.267 (22) 0.333 (23) 0.266 (24)
0.333 (25) 0.267 (26) 0.333 (27) 0.267 (28) 0.333 (29)
Average = 0.300 Range = 0.067

rule 21
0.483 (0) 0.483 (1) 0.483 (2) 0.483 (3) 0.483 (4)
0.483 (5) 0.483 (6) 0.483 (7) 0.483 (8) 0.483 (9)
0.483 (10) 0.484 (11) 0.483 (12) 0.483 (13) 0.483 (14)
0.483 (15) 0.483 (16) 0.483 (17) 0.483 (18) 0.483 (19)
0.483 (20) 0.483 (21) 0.483 (22) 0.484 (23) 0.483 (24)
0.484 (25) 0.483 (26) 0.483 (27) 0.483 (28) 0.483 (29)
Average = 0.483 Range = 0.000

rule 22
0.003 (0) 0.002 (1) 0.002 (2) 0.002 (3) 0.003 (4)
0.003 (5) 0.003 (6) 0.003 (7) 0.003 (8) 0.003 (9)
0.002 (10) 0.003 (11) 0.002 (12) 0.002 (13) 0.002 (14)
0.003 (15) 0.003 (16) 0.002 (17) 0.002 (18) 0.002 (19)
0.003 (20) 0.003 (21) 0.003 (22) 0.003 (23) 0.003 (24)
0.002 (25) 0.002 (26) 0.003 (27) 0.003 (28) 0.003 (29)
Average = 0.002 Range = 0.001

rule 23
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.500 (28) 0.500 (29)
Average = 0.500 Range = 0.000

rule 24
0.167 (0) 0.167 (1) 0.167 (2) 0.167 (3) 0.167 (4)
0.167 (5) 0.167 (6) 0.167 (7) 0.167 (8) 0.167 (9)
0.167 (10) 0.167 (11) 0.167 (12) 0.167 (13) 0.167 (14)
0.167 (15) 0.167 (16) 0.167 (17) 0.167 (18) 0.167 (19)
0.167 (20) 0.167 (21) 0.167 (22) 0.167 (23) 0.167 (24)
0.167 (25) 0.167 (26) 0.167 (27) 0.167 (28) 0.167 (29)
Average = 0.167 Range = 0.000

rule 25
0.450 (0) 0.450 (1) 0.450 (2) 0.450 (3) 0.450 (4)
0.450 (5) 0.450 (6) 0.450 (7) 0.450 (8) 0.450 (9)
0.450 (10) 0.450 (11) 0.450 (12) 0.450 (13) 0.450 (14)
0.450 (15) 0.450 (16) 0.450 (17) 0.450 (18) 0.450 (19)
0.450 (20) 0.450 (21) 0.450 (22) 0.450 (23) 0.450 (24)
0.450 (25) 0.450 (26) 0.450 (27) 0.450 (28) 0.450 (29)
Average = 0.450 Range = 0.000

rule 26
0.367 (0) 0.450 (1) 0.367 (2) 0.450 (3) 0.366 (4)
0.450 (5) 0.367 (6) 0.450 (7) 0.367 (8) 0.450 (9)
0.367 (10) 0.450 (11) 0.366 (12) 0.450 (13) 0.367 (14)
0.450 (15) 0.367 (16) 0.450 (17) 0.367 (18) 0.450 (19)
0.367 (20) 0.450 (21) 0.367 (22) 0.450 (23) 0.367 (24)
0.450 (25) 0.367 (26) 0.450 (27) 0.367 (28) 0.450 (29)
Average = 0.408 Range = 0.084

rule 27
0.533 (0) 0.533 (1) 0.533 (2) 0.533 (3) 0.533 (4)
0.533 (5) 0.533 (6) 0.533 (7) 0.533 (8) 0.533 (9)
0.533 (10) 0.533 (11) 0.533 (12) 0.533 (13) 0.533 (14)
0.533 (15) 0.533 (16) 0.533 (17) 0.533 (18) 0.533 (19)
0.533 (20) 0.533 (21) 0.533 (22) 0.533 (23) 0.533 (24)
0.533 (25) 0.533 (26) 0.533 (27) 0.533 (28) 0.533 (29)
Average = 0.533 Range = 0.000

rule 28
0.500 (0) 0.000 (1) 1.000 (2) 0.000 (3) 1.000 (4)
0.500 (5) 0.000 (6) 1.000 (7) 0.500 (8) 0.000 (9)
1.000 (10) 0.000 (11) 1.000 (12) 0.000 (13) 1.000 (14)
0.500 (15) 0.000 (16) 1.000 (17) 0.000 (18) 1.000 (19)
0.000 (20) 1.000 (21) 0.500 (22) 0.000 (23) 1.000 (24)
0.000 (25) 1.000 (26) 0.500 (27) 0.000 (28) 1.000 (29)
Average = 0.500 Range = 1.000

rule 29
0.000 (0) 1.000 (1) 0.500 (2) 0.000 (3) 1.000 (4)
0.000 (5) 1.000 (6) 0.000 (7) 1.000 (8) 0.500 (9)
0.000 (10) 1.000 (11) 0.000 (12) 1.000 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.000 (17) 1.000 (18) 0.500 (19)
0.000 (20) 1.000 (21) 0.000 (22) 1.000 (23) 0.000 (24)
1.000 (25) 0.000 (26) 1.000 (27) 0.000 (28) 1.000 (29)
Average = 0.500 Range = 1.000

rule 30
0.502 (0) 0.506 (1) 0.500 (2) 0.504 (3) 0.486 (4)
0.504 (5) 0.494 (6) 0.501 (7) 0.496 (8) 0.505 (9)
0.495 (10) 0.496 (11) 0.497 (12) 0.493 (13) 0.507 (14)
0.498 (15) 0.507 (16) 0.494 (17) 0.503 (18) 0.496 (19)
0.505 (20) 0.491 (21) 0.509 (22) 0.494 (23) 0.502 (24)
0.503 (25) 0.497 (26) 0.499 (27) 0.500 (28) 0.499 (29)
Average = 0.499 Range = 0.024

rule 31
0.550 (0) 0.550 (1) 0.550 (2) 0.550 (3) 0.550 (4)
0.550 (5) 0.550 (6) 0.550 (7) 0.550 (8) 0.550 (9)
0.550 (10) 0.550 (11) 0.550 (12) 0.550 (13) 0.550 (14)
0.550 (15) 0.550 (16) 0.550 (17) 0.550 (18) 0.550 (19)
0.550 (20) 0.550 (21) 0.550 (22) 0.550 (23) 0.550 (24)
0.550 (25) 0.550 (26) 0.550 (27) 0.550 (28) 0.550 (29)
Average = 0.550 Range = 0.001

rule 32
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 0.000 (4)
0.000 (5) 0.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
0.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 0.000 (29)
Average = 0.000 Range = 0.000

rule 33
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.000 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.000 (10) 0.500 (11) 0.000 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.000 (17) 0.500 (18) 0.000 (19)
0.500 (20) 0.000 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.000 (25) 0.500 (26) 0.000 (27) 0.500 (28) 0.500 (29)
Average = 0.367 Range = 0.500

rule 34
0.300 (0) 0.300 (1) 0.300 (2) 0.300 (3) 0.300 (4)
0.300 (5) 0.300 (6) 0.300 (7) 0.300 (8) 0.300 (9)
0.300 (10) 0.300 (11) 0.300 (12) 0.300 (13) 0.300 (14)
0.300 (15) 0.300 (16) 0.300 (17) 0.300 (18) 0.300 (19)
0.300 (20) 0.300 (21) 0.300 (22) 0.300 (23) 0.300 (24)
0.300 (25) 0.300 (26) 0.300 (27) 0.300 (28) 0.300 (29)
Average = 0.300 Range = 0.000

rule 35
0.367 (0) 0.367 (1) 0.367 (2) 0.367 (3) 0.367 (4)
0.367 (5) 0.367 (6) 0.367 (7) 0.367 (8) 0.367 (9)
0.367 (10) 0.367 (11) 0.367 (12) 0.367 (13) 0.367 (14)
0.367 (15) 0.367 (16) 0.367 (17) 0.367 (18) 0.367 (19)
0.367 (20) 0.367 (21) 0.367 (22) 0.367 (23) 0.367 (24)
0.367 (25) 0.367 (26) 0.367 (27) 0.367 (28) 0.367 (29)
Average = 0.367 Range = 0.000

rule 36
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 0.000 (4)
0.000 (5) 0.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
1.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
0.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 0.000 (29)
Average = 0.033 Range = 1.000

rule 37
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.000 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.000 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.500 (17) 0.000 (18) 0.500 (19)
0.500 (20) 0.000 (21) 0.500 (22) 0.500 (23) 0.000 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.001 (28) 0.500 (29)
Average = 0.400 Range = 0.500

rule 38
0.333 (0) 0.367 (1) 0.333 (2) 0.367 (3) 0.333 (4)
0.367 (5) 0.333 (6) 0.367 (7) 0.333 (8) 0.367 (9)
0.333 (10) 0.367 (11) 0.333 (12) 0.367 (13) 0.333 (14)
0.367 (15) 0.333 (16) 0.367 (17) 0.333 (18) 0.367 (19)
0.333 (20) 0.367 (21) 0.333 (22) 0.367 (23) 0.333 (24)
0.367 (25) 0.333 (26) 0.367 (27) 0.333 (28) 0.367 (29)
Average = 0.350 Range = 0.034

rule 39
0.467 (0) 0.467 (1) 0.467 (2) 0.467 (3) 0.467 (4)
0.467 (5) 0.467 (6) 0.467 (7) 0.467 (8) 0.467 (9)
0.467 (10) 0.467 (11) 0.467 (12) 0.467 (13) 0.467 (14)
0.467 (15) 0.467 (16) 0.467 (17) 0.467 (18) 0.467 (19)
0.467 (20) 0.467 (21) 0.467 (22) 0.467 (23) 0.467 (24)
0.467 (25) 0.467 (26) 0.467 (27) 0.467 (28) 0.467 (29)
Average = 0.467 Range = 0.000

rule 40
0.000 (0) 0.001 (1) 0.001 (2) 0.001 (3) 0.001 (4)
0.001 (5) 0.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
0.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 0.000 (29)
Average = 0.000 Range = 0.001

rule 41
0.365 (0) 0.365 (1) 0.364 (2) 0.365 (3) 0.365 (4)
0.365 (5) 0.364 (6) 0.365 (7) 0.365 (8) 0.365 (9)
0.364 (10) 0.365 (11) 0.366 (12) 0.365 (13) 0.364 (14)
0.365 (15) 0.366 (16) 0.365 (17) 0.365 (18) 0.365 (19)
0.365 (20) 0.365 (21) 0.365 (22) 0.365 (23) 0.365 (24)
0.365 (25) 0.365 (26) 0.365 (27) 0.365 (28) 0.365 (29)
Average = 0.365 Range = 0.002

rule 42
0.467 (0) 0.467 (1) 0.467 (2) 0.467 (3) 0.467 (4)
0.467 (5) 0.467 (6) 0.467 (7) 0.467 (8) 0.467 (9)
0.467 (10) 0.467 (11) 0.467 (12) 0.467 (13) 0.467 (14)
0.467 (15) 0.467 (16) 0.467 (17) 0.467 (18) 0.467 (19)
0.467 (20) 0.467 (21) 0.467 (22) 0.467 (23) 0.467 (24)
0.467 (25) 0.467 (26) 0.467 (27) 0.467 (28) 0.467 (29)
Average = 0.467 Range = 0.000

rule 43
0.533 (0) 0.533 (1) 0.533 (2) 0.533 (3) 0.533 (4)
0.533 (5) 0.533 (6) 0.533 (7) 0.533 (8) 0.533 (9)
0.533 (10) 0.533 (11) 0.533 (12) 0.533 (13) 0.533 (14)
0.533 (15) 0.533 (16) 0.533 (17) 0.533 (18) 0.534 (19)
0.533 (20) 0.533 (21) 0.533 (22) 0.533 (23) 0.533 (24)
0.533 (25) 0.533 (26) 0.533 (27) 0.533 (28) 0.533 (29)
Average = 0.533 Range = 0.000

rule 44
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 0.000 (4)
1.000 (5) 0.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 1.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 1.000 (21) 0.000 (22) 0.000 (23) 1.000 (24)
0.000 (25) 0.000 (26) 0.000 (27) 1.000 (28) 0.000 (29)
Average = 0.167 Range = 1.000

rule 45
0.500 (0) 0.500 (1) 0.498 (2) 0.499 (3) 0.497 (4)
0.509 (5) 0.497 (6) 0.505 (7) 0.494 (8) 0.505 (9)
0.499 (10) 0.501 (11) 0.498 (12) 0.508 (13) 0.491 (14)
0.503 (15) 0.507 (16) 0.494 (17) 0.504 (18) 0.501 (19)
0.496 (20) 0.502 (21) 0.503 (22) 0.507 (23) 0.497 (24)
0.496 (25) 0.500 (26) 0.507 (27) 0.495 (28) 0.507 (29)
Average = 0.501 Range = 0.018

rule 46
0.400 (0) 0.400 (1) 0.400 (2) 0.400 (3) 0.400 (4)
0.400 (5) 0.400 (6) 0.400 (7) 0.400 (8) 0.400 (9)
0.400 (10) 0.400 (11) 0.400 (12) 0.400 (13) 0.400 (14)
0.400 (15) 0.400 (16) 0.400 (17) 0.400 (18) 0.400 (19)
0.400 (20) 0.400 (21) 0.400 (22) 0.400 (23) 0.400 (24)
0.400 (25) 0.400 (26) 0.400 (27) 0.400 (28) 0.400 (29)
Average = 0.400 Range = 0.000

rule 47
0.467 (0) 0.533 (1) 0.467 (2) 0.533 (3) 0.467 (4)
0.533 (5) 0.467 (6) 0.533 (7) 0.467 (8) 0.534 (9)
0.467 (10) 0.533 (11) 0.467 (12) 0.533 (13) 0.467 (14)
0.533 (15) 0.467 (16) 0.533 (17) 0.467 (18) 0.533 (19)
0.467 (20) 0.533 (21) 0.467 (22) 0.533 (23) 0.467 (24)
0.533 (25) 0.467 (26) 0.533 (27) 0.467 (28) 0.533 (29)
Average = 0.500 Range = 0.067

rule 48
0.300 (0) 0.300 (1) 0.300 (2) 0.300 (3) 0.300 (4)
0.300 (5) 0.300 (6) 0.300 (7) 0.300 (8) 0.300 (9)
0.300 (10) 0.300 (11) 0.300 (12) 0.300 (13) 0.300 (14)
0.300 (15) 0.300 (16) 0.300 (17) 0.300 (18) 0.300 (19)
0.300 (20) 0.300 (21) 0.300 (22) 0.300 (23) 0.300 (24)
0.300 (25) 0.300 (26) 0.300 (27) 0.300 (28) 0.300 (29)
Average = 0.300 Range = 0.000

rule 49
0.383 (0) 0.383 (1) 0.383 (2) 0.383 (3) 0.383 (4)
0.383 (5) 0.383 (6) 0.383 (7) 0.383 (8) 0.383 (9)
0.383 (10) 0.383 (11) 0.383 (12) 0.383 (13) 0.383 (14)
0.383 (15) 0.383 (16) 0.383 (17) 0.383 (18) 0.383 (19)
0.383 (20) 0.383 (21) 0.383 (22) 0.383 (23) 0.383 (24)
0.383 (25) 0.383 (26) 0.383 (27) 0.383 (28) 0.383 (29)
Average = 0.383 Range = 0.000

rule 50
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.500 (28) 0.500 (29)
Average = 0.500 Range = 0.000

rule 51
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.500 (28) 0.500 (29)
Average = 0.500 Range = 0.000

rule 52
0.333 (0) 0.367 (1) 0.333 (2) 0.367 (3) 0.333 (4)
0.367 (5) 0.333 (6) 0.367 (7) 0.333 (8) 0.367 (9)
0.333 (10) 0.367 (11) 0.333 (12) 0.367 (13) 0.333 (14)
0.367 (15) 0.333 (16) 0.367 (17) 0.333 (18) 0.367 (19)
0.333 (20) 0.367 (21) 0.333 (22) 0.367 (23) 0.333 (24)
0.367 (25) 0.333 (26) 0.367 (27) 0.333 (28) 0.367 (29)
Average = 0.350 Range = 0.034

rule 53
0.483 (0) 0.483 (1) 0.483 (2) 0.483 (3) 0.483 (4)
0.483 (5) 0.483 (6) 0.483 (7) 0.483 (8) 0.483 (9)
0.483 (10) 0.483 (11) 0.483 (12) 0.483 (13) 0.483 (14)
0.483 (15) 0.483 (16) 0.483 (17) 0.483 (18) 0.483 (19)
0.483 (20) 0.483 (21) 0.483 (22) 0.483 (23) 0.483 (24)
0.483 (25) 0.483 (26) 0.483 (27) 0.483 (28) 0.483 (29)
Average = 0.483 Range = 0.000

rule 54
0.500 (0) 0.250 (1) 0.250 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.250 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.500 (17) 0.250 (18) 0.250 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.250 (25) 0.500 (26) 0.500 (27) 0.500 (28) 0.500 (29)
Average = 0.450 Range = 0.250

rule 55
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.500 (28) 0.500 (29)
Average = 0.500 Range = 0.000

rule 56
0.300 (0) 0.300 (1) 0.300 (2) 0.300 (3) 0.300 (4)
0.300 (5) 0.300 (6) 0.300 (7) 0.300 (8) 0.300 (9)
0.300 (10) 0.300 (11) 0.300 (12) 0.300 (13) 0.300 (14)
0.300 (15) 0.300 (16) 0.300 (17) 0.300 (18) 0.300 (19)
0.300 (20) 0.300 (21) 0.300 (22) 0.300 (23) 0.300 (24)
0.300 (25) 0.300 (26) 0.300 (27) 0.300 (28) 0.300 (29)
Average = 0.300 Range = 0.000

rule 57
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.500 (28) 0.500 (29)
Average = 0.500 Range = 0.000

rule 58
0.667 (0) 0.667 (1) 0.667 (2) 0.667 (3) 0.667 (4)
0.667 (5) 0.667 (6) 0.667 (7) 0.667 (8) 0.667 (9)
0.667 (10) 0.667 (11) 0.667 (12) 0.667 (13) 0.667 (14)
0.667 (15) 0.667 (16) 0.667 (17) 0.666 (18) 0.666 (19)
0.667 (20) 0.667 (21) 0.667 (22) 0.667 (23) 0.667 (24)
0.667 (25) 0.667 (26) 0.667 (27) 0.667 (28) 0.667 (29)
Average = 0.667 Range = 0.000

rule 59
0.617 (0) 0.617 (1) 0.617 (2) 0.617 (3) 0.617 (4)
0.617 (5) 0.617 (6) 0.617 (7) 0.617 (8) 0.617 (9)
0.617 (10) 0.617 (11) 0.617 (12) 0.617 (13) 0.617 (14)
0.617 (15) 0.617 (16) 0.617 (17) 0.617 (18) 0.617 (19)
0.617 (20) 0.617 (21) 0.617 (22) 0.617 (23) 0.617 (24)
0.617 (25) 0.617 (26) 0.617 (27) 0.617 (28) 0.617 (29)
Average = 0.617 Range = 0.000

rule 60
0.533 (0) 0.533 (1) 0.467 (2) 0.600 (3) 0.533 (4)
0.533 (5) 0.467 (6) 0.400 (7) 0.600 (8) 0.533 (9)
0.467 (10) 0.467 (11) 0.400 (12) 0.467 (13) 0.533 (14)
0.600 (15) 0.533 (16) 0.600 (17) 0.534 (18) 0.467 (19)
0.400 (20) 0.600 (21) 0.533 (22) 0.400 (23) 0.600 (24)
0.600 (25) 0.533 (26) 0.333 (27) 0.667 (28) 0.600 (29)
Average = 0.518 Range = 0.333

rule 61
0.567 (0) 0.567 (1) 0.567 (2) 0.567 (3) 0.567 (4)
0.567 (5) 0.567 (6) 0.567 (7) 0.567 (8) 0.567 (9)
0.567 (10) 0.567 (11) 0.567 (12) 0.567 (13) 0.567 (14)
0.567 (15) 0.567 (16) 0.567 (17) 0.567 (18) 0.567 (19)
0.567 (20) 0.567 (21) 0.567 (22) 0.567 (23) 0.567 (24)
0.567 (25) 0.567 (26) 0.567 (27) 0.567 (28) 0.567 (29)
Average = 0.567 Range = 0.000

rule 62
0.667 (0) 0.667 (1) 0.667 (2) 0.667 (3) 0.333 (4)
0.667 (5) 0.667 (6) 0.667 (7) 0.333 (8) 0.667 (9)
0.667 (10) 0.333 (11) 0.667 (12) 0.667 (13) 0.667 (14)
0.667 (15) 0.667 (16) 0.667 (17) 0.667 (18) 0.667 (19)
0.667 (20) 0.667 (21) 0.667 (22) 0.666 (23) 0.666 (24)
0.334 (25) 0.334 (26) 0.666 (27) 0.666 (28) 0.666 (29)
Average = 0.611 Range = 0.333

rule 63
0.533 (0) 0.533 (1) 0.533 (2) 0.533 (3) 0.534 (4)
0.533 (5) 0.533 (6) 0.533 (7) 0.533 (8) 0.533 (9)
0.533 (10) 0.533 (11) 0.533 (12) 0.533 (13) 0.533 (14)
0.533 (15) 0.533 (16) 0.533 (17) 0.533 (18) 0.533 (19)
0.533 (20) 0.533 (21) 0.533 (22) 0.533 (23) 0.533 (24)
0.533 (25) 0.533 (26) 0.533 (27) 0.533 (28) 0.534 (29)
Average = 0.533 Range = 0.000

rule 64
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 0.000 (4)
0.000 (5) 0.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
0.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 0.000 (29)
Average = 0.000 Range = 0.000

rule 65
0.534 (0) 0.266 (1) 0.533 (2) 0.267 (3) 0.533 (4)
0.267 (5) 0.533 (6) 0.267 (7) 0.533 (8) 0.267 (9)
0.533 (10) 0.267 (11) 0.533 (12) 0.266 (13) 0.533 (14)
0.267 (15) 0.533 (16) 0.267 (17) 0.533 (18) 0.267 (19)
0.533 (20) 0.267 (21) 0.533 (22) 0.267 (23) 0.533 (24)
0.267 (25) 0.533 (26) 0.267 (27) 0.533 (28) 0.267 (29)
Average = 0.400 Range = 0.267

rule 66
0.200 (0) 0.200 (1) 0.200 (2) 0.200 (3) 0.200 (4)
0.200 (5) 0.200 (6) 0.200 (7) 0.200 (8) 0.200 (9)
0.200 (10) 0.200 (11) 0.200 (12) 0.200 (13) 0.200 (14)
0.200 (15) 0.200 (16) 0.200 (17) 0.200 (18) 0.200 (19)
0.200 (20) 0.200 (21) 0.200 (22) 0.200 (23) 0.200 (24)
0.200 (25) 0.200 (26) 0.200 (27) 0.200 (28) 0.200 (29)
Average = 0.200 Range = 0.000

rule 67
0.450 (0) 0.450 (1) 0.450 (2) 0.450 (3) 0.450 (4)
0.450 (5) 0.450 (6) 0.450 (7) 0.450 (8) 0.450 (9)
0.450 (10) 0.450 (11) 0.450 (12) 0.450 (13) 0.450 (14)
0.450 (15) 0.450 (16) 0.450 (17) 0.450 (18) 0.450 (19)
0.450 (20) 0.450 (21) 0.450 (22) 0.450 (23) 0.450 (24)
0.450 (25) 0.450 (26) 0.450 (27) 0.450 (28) 0.450 (29)
Average = 0.450 Range = 0.000

rule 68
0.000 (0) 0.000 (1) 0.000 (2) 1.000 (3) 0.000 (4)
0.000 (5) 1.000 (6) 0.000 (7) 0.000 (8) 1.000 (9)
0.000 (10) 1.000 (11) 0.000 (12) 1.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 1.000 (19)
0.000 (20) 0.000 (21) 1.000 (22) 0.000 (23) 0.000 (24)
1.000 (25) 0.000 (26) 0.000 (27) 1.000 (28) 0.000 (29)
Average = 0.300 Range = 1.000

rule 69
1.000 (0) 0.000 (1) 1.000 (2) 0.000 (3) 0.000 (4)
1.000 (5) 0.000 (6) 0.000 (7) 1.000 (8) 0.000 (9)
1.000 (10) 0.000 (11) 1.000 (12) 0.000 (13) 1.000 (14)
0.000 (15) 1.000 (16) 0.000 (17) 0.000 (18) 1.000 (19)
0.000 (20) 1.000 (21) 0.000 (22) 1.000 (23) 0.000 (24)
1.000 (25) 0.000 (26) 1.000 (27) 0.000 (28) 0.000 (29)
Average = 0.433 Range = 1.000

rule 70
0.500 (0) 1.000 (1) 0.000 (2) 1.000 (3) 0.000 (4)
0.500 (5) 1.000 (6) 0.000 (7) 1.000 (8) 0.000 (9)
1.000 (10) 0.000 (11) 1.000 (12) 0.000 (13) 1.000 (14)
0.000 (15) 0.500 (16) 1.000 (17) 0.000 (18) 1.000 (19)
0.000 (20) 0.500 (21) 1.000 (22) 0.000 (23) 1.000 (24)
0.000 (25) 1.000 (26) 0.000 (27) 1.000 (28) 0.000 (29)
Average = 0.500 Range = 1.000

rule 71
0.000 (0) 0.500 (1) 0.500 (2) 0.500 (3) 1.000 (4)
0.000 (5) 0.500 (6) 1.000 (7) 0.000 (8) 1.000 (9)
0.000 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.500 (14)
1.000 (15) 0.000 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 1.000 (21) 0.000 (22) 1.000 (23) 0.000 (24)
1.000 (25) 0.000 (26) 0.500 (27) 0.500 (28) 1.000 (29)
Average = 0.500 Range = 1.000

rule 72
1.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 1.000 (4)
1.000 (5) 0.000 (6) 1.000 (7) 1.000 (8) 0.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
0.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 1.000 (29)
Average = 0.200 Range = 1.000

rule 73
1.000 (0) 1.000 (1) 0.000 (2) 0.500 (3) 0.000 (4)
1.000 (5) 1.000 (6) 0.000 (7) 0.583 (8) 0.167 (9)
0.583 (10) 0.167 (11) 0.583 (12) 0.167 (13) 0.583 (14)
0.000 (15) 1.000 (16) 1.000 (17) 0.000 (18) 0.500 (19)
0.000 (20) 0.500 (21) 0.000 (22) 0.500 (23) 0.000 (24)
1.000 (25) 1.000 (26) 0.000 (27) 0.500 (28) 0.000 (29)
Average = 0.444 Range = 1.000

rule 74
0.200 (0) 0.200 (1) 0.200 (2) 0.200 (3) 0.200 (4)
0.200 (5) 0.200 (6) 0.200 (7) 0.200 (8) 0.200 (9)
0.200 (10) 0.200 (11) 0.200 (12) 0.200 (13) 0.200 (14)
0.200 (15) 0.200 (16) 0.200 (17) 0.200 (18) 0.200 (19)
0.200 (20) 0.200 (21) 0.200 (22) 0.200 (23) 0.200 (24)
0.200 (25) 0.200 (26) 0.200 (27) 0.200 (28) 0.200 (29)
Average = 0.200 Range = 0.000

rule 75
0.508 (0) 0.495 (1) 0.502 (2) 0.499 (3) 0.506 (4)
0.497 (5) 0.500 (6) 0.494 (7) 0.504 (8) 0.498 (9)
0.498 (10) 0.504 (11) 0.502 (12) 0.500 (13) 0.506 (14)
0.492 (15) 0.503 (16) 0.499 (17) 0.502 (18) 0.507 (19)
0.496 (20) 0.498 (21) 0.497 (22) 0.501 (23) 0.494 (24)
0.499 (25) 0.500 (26) 0.502 (27) 0.496 (28) 0.502 (29)
Average = 0.500 Range = 0.015

rule 76
1.000 (0) 0.000 (1) 1.000 (2) 0.000 (3) 1.000 (4)
1.000 (5) 0.000 (6) 1.000 (7) 1.000 (8) 0.000 (9)
1.000 (10) 0.000 (11) 1.000 (12) 0.000 (13) 1.000 (14)
0.000 (15) 1.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 1.000 (21) 0.000 (22) 0.000 (23) 1.000 (24)
0.000 (25) 0.000 (26) 1.000 (27) 0.000 (28) 1.000 (29)
Average = 0.467 Range = 1.000

rule 77
1.000 (0) 1.000 (1) 0.000 (2) 1.000 (3) 0.000 (4)
1.000 (5) 1.000 (6) 0.000 (7) 1.000 (8) 0.000 (9)
1.000 (10) 0.000 (11) 1.000 (12) 0.000 (13) 1.000 (14)
0.000 (15) 1.000 (16) 1.000 (17) 0.000 (18) 0.000 (19)
1.000 (20) 0.000 (21) 1.000 (22) 0.000 (23) 0.000 (24)
1.000 (25) 1.000 (26) 0.000 (27) 1.000 (28) 0.000 (29)
Average = 0.533 Range = 1.000

rule 78
1.000 (0) 1.000 (1) 0.000 (2) 1.000 (3) 0.000 (4)
1.000 (5) 1.000 (6) 0.000 (7) 1.000 (8) 1.000 (9)
0.000 (10) 1.000 (11) 0.000 (12) 1.000 (13) 0.000 (14)
1.000 (15) 0.000 (16) 1.000 (17) 0.000 (18) 1.000 (19)
0.000 (20) 1.000 (21) 1.000 (22) 0.000 (23) 1.000 (24)
1.000 (25) 0.000 (26) 1.000 (27) 1.000 (28) 0.000 (29)
Average = 0.600 Range = 1.000

rule 79
1.000 (0) 0.000 (1) 1.000 (2) 0.000 (3) 1.000 (4)
1.000 (5) 0.000 (6) 1.000 (7) 1.000 (8) 0.000 (9)
1.000 (10) 0.000 (11) 1.000 (12) 0.000 (13) 1.000 (14)
0.000 (15) 1.000 (16) 0.000 (17) 1.000 (18) 0.000 (19)
1.000 (20) 1.000 (21) 0.000 (22) 1.000 (23) 1.000 (24)
0.000 (25) 1.000 (26) 1.000 (27) 0.000 (28) 1.000 (29)
Average = 0.600 Range = 1.000

rule 80
0.300 (0) 0.300 (1) 0.300 (2) 0.300 (3) 0.300 (4)
0.300 (5) 0.300 (6) 0.300 (7) 0.300 (8) 0.300 (9)
0.300 (10) 0.300 (11) 0.300 (12) 0.300 (13) 0.300 (14)
0.300 (15) 0.300 (16) 0.300 (17) 0.300 (18) 0.300 (19)
0.300 (20) 0.300 (21) 0.300 (22) 0.300 (23) 0.300 (24)
0.300 (25) 0.300 (26) 0.300 (27) 0.300 (28) 0.300 (29)
Average = 0.300 Range = 0.000

rule 81
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.500 (28) 0.500 (29)
Average = 0.500 Range = 0.000

rule 82
0.350 (0) 0.433 (1) 0.350 (2) 0.433 (3) 0.350 (4)
0.433 (5) 0.350 (6) 0.433 (7) 0.350 (8) 0.433 (9)
0.350 (10) 0.433 (11) 0.350 (12) 0.433 (13) 0.350 (14)
0.433 (15) 0.350 (16) 0.433 (17) 0.350 (18) 0.433 (19)
0.350 (20) 0.433 (21) 0.350 (22) 0.433 (23) 0.350 (24)
0.433 (25) 0.350 (26) 0.433 (27) 0.350 (28) 0.433 (29)
Average = 0.392 Range = 0.084

rule 83
0.550 (0) 0.550 (1) 0.550 (2) 0.550 (3) 0.550 (4)
0.550 (5) 0.550 (6) 0.550 (7) 0.550 (8) 0.550 (9)
0.550 (10) 0.550 (11) 0.550 (12) 0.550 (13) 0.550 (14)
0.550 (15) 0.550 (16) 0.550 (17) 0.550 (18) 0.550 (19)
0.550 (20) 0.550 (21) 0.550 (22) 0.550 (23) 0.550 (24)
0.550 (25) 0.550 (26) 0.550 (27) 0.550 (28) 0.550 (29)
Average = 0.550 Range = 0.000

rule 84
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.500 (28) 0.500 (29)
Average = 0.500 Range = 0.000

rule 85
0.367 (0) 0.633 (1) 0.367 (2) 0.633 (3) 0.367 (4)
0.633 (5) 0.366 (6) 0.633 (7) 0.366 (8) 0.633 (9)
0.367 (10) 0.633 (11) 0.367 (12) 0.633 (13) 0.367 (14)
0.633 (15) 0.367 (16) 0.633 (17) 0.367 (18) 0.633 (19)
0.367 (20) 0.633 (21) 0.367 (22) 0.633 (23) 0.367 (24)
0.633 (25) 0.367 (26) 0.633 (27) 0.367 (28) 0.633 (29)
Average = 0.500 Range = 0.267

rule 86
0.498 (0) 0.502 (1) 0.500 (2) 0.505 (3) 0.497 (4)
0.500 (5) 0.507 (6) 0.500 (7) 0.501 (8) 0.502 (9)
0.495 (10) 0.501 (11) 0.495 (12) 0.505 (13) 0.493 (14)
0.506 (15) 0.497 (16) 0.496 (17) 0.496 (18) 0.513 (19)
0.486 (20) 0.509 (21) 0.493 (22) 0.505 (23) 0.499 (24)
0.499 (25) 0.492 (26) 0.505 (27) 0.502 (28) 0.503 (29)
Average = 0.500 Range = 0.027

rule 87
0.517 (0) 0.516 (1) 0.517 (2) 0.517 (3) 0.517 (4)
0.517 (5) 0.517 (6) 0.517 (7) 0.517 (8) 0.517 (9)
0.517 (10) 0.517 (11) 0.517 (12) 0.517 (13) 0.517 (14)
0.517 (15) 0.517 (16) 0.517 (17) 0.517 (18) 0.517 (19)
0.517 (20) 0.517 (21) 0.517 (22) 0.517 (23) 0.517 (24)
0.517 (25) 0.517 (26) 0.517 (27) 0.517 (28) 0.517 (29)
Average = 0.517 Range = 0.000

rule 88
0.400 (0) 0.400 (1) 0.400 (2) 0.400 (3) 0.400 (4)
0.400 (5) 0.400 (6) 0.400 (7) 0.400 (8) 0.400 (9)
0.400 (10) 0.400 (11) 0.400 (12) 0.400 (13) 0.400 (14)
0.400 (15) 0.400 (16) 0.400 (17) 0.400 (18) 0.400 (19)
0.400 (20) 0.400 (21) 0.400 (22) 0.400 (23) 0.400 (24)
0.400 (25) 0.400 (26) 0.400 (27) 0.400 (28) 0.400 (29)
Average = 0.400 Range = 0.001

rule 89
0.505 (0) 0.497 (1) 0.505 (2) 0.509 (3) 0.489 (4)
0.507 (5) 0.504 (6) 0.489 (7) 0.508 (8) 0.505 (9)
0.498 (10) 0.498 (11) 0.498 (12) 0.503 (13) 0.505 (14)
0.490 (15) 0.500 (16) 0.511 (17) 0.494 (18) 0.501 (19)
0.505 (20) 0.493 (21) 0.506 (22) 0.496 (23) 0.500 (24)
0.504 (25) 0.495 (26) 0.506 (27) 0.502 (28) 0.501 (29)
Average = 0.501 Range = 0.022

rule 90
0.633 (0) 0.433 (1) 0.467 (2) 0.500 (3) 0.500 (4)
0.400 (5) 0.567 (6) 0.567 (7) 0.667 (8) 0.633 (9)
0.500 (10) 0.200 (11) 0.567 (12) 0.633 (13) 0.533 (14)
0.633 (15) 0.433 (16) 0.467 (17) 0.500 (18) 0.500 (19)
0.400 (20) 0.567 (21) 0.567 (22) 0.666 (23) 0.633 (24)
0.500 (25) 0.200 (26) 0.567 (27) 0.633 (28) 0.533 (29)
Average = 0.520 Range = 0.466

rule 91
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.999 (6) 0.500 (7) 0.500 (8) 0.999 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 1.000 (14)
0.500 (15) 0.500 (16) 1.000 (17) 0.500 (18) 0.500 (19)
0.999 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.999 (28) 0.500 (29)
Average = 0.600 Range = 0.500

rule 92
1.000 (0) 1.000 (1) 0.000 (2) 1.000 (3) 0.000 (4)
1.000 (5) 0.000 (6) 1.000 (7) 1.000 (8) 0.000 (9)
1.000 (10) 0.000 (11) 1.000 (12) 0.000 (13) 1.000 (14)
0.000 (15) 1.000 (16) 0.000 (17) 1.000 (18) 0.000 (19)
1.000 (20) 0.000 (21) 1.000 (22) 1.000 (23) 0.000 (24)
1.000 (25) 1.000 (26) 0.000 (27) 1.000 (28) 0.000 (29)
Average = 0.567 Range = 1.000

rule 93
1.000 (0) 0.000 (1) 1.000 (2) 0.000 (3) 1.000 (4)
1.000 (5) 0.000 (6) 1.000 (7) 0.000 (8) 1.000 (9)
0.000 (10) 1.000 (11) 0.000 (12) 1.000 (13) 0.000 (14)
1.000 (15) 0.000 (16) 1.000 (17) 0.000 (18) 1.000 (19)
0.000 (20) 1.000 (21) 1.000 (22) 0.000 (23) 1.000 (24)
1.000 (25) 0.000 (26) 1.000 (27) 0.000 (28) 1.000 (29)
Average = 0.567 Range = 1.000

rule 94
1.000 (0) 1.000 (1) 0.000 (2) 1.000 (3) 0.500 (4)
0.500 (5) 1.000 (6) 0.000 (7) 1.000 (8) 1.000 (9)
0.000 (10) 1.000 (11) 0.000 (12) 1.000 (13) 1.000 (14)
0.000 (15) 1.000 (16) 1.000 (17) 0.000 (18) 1.000 (19)
0.000 (20) 1.000 (21) 1.000 (22) 0.000 (23) 1.000 (24)
1.000 (25) 0.000 (26) 1.000 (27) 1.000 (28) 0.000 (29)
Average = 0.633 Range = 1.000

rule 95
1.000 (0) 0.000 (1) 1.000 (2) 0.000 (3) 1.000 (4)
1.000 (5) 0.000 (6) 1.000 (7) 1.000 (8) 0.000 (9)
1.000 (10) 0.000 (11) 1.000 (12) 0.000 (13) 1.000 (14)
0.000 (15) 1.000 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 1.000 (24)
0.500 (25) 0.500 (26) 1.000 (27) 0.000 (28) 1.000 (29)
Average = 0.583 Range = 1.000

rule 96
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 0.000 (4)
0.000 (5) 0.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
0.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 0.000 (29)
Average = 0.000 Range = 0.000

rule 97
0.364 (0) 0.365 (1) 0.364 (2) 0.366 (3) 0.363 (4)
0.366 (5) 0.365 (6) 0.365 (7) 0.365 (8) 0.365 (9)
0.365 (10) 0.366 (11) 0.365 (12) 0.364 (13) 0.365 (14)
0.365 (15) 0.365 (16) 0.364 (17) 0.365 (18) 0.365 (19)
0.366 (20) 0.364 (21) 0.365 (22) 0.364 (23) 0.366 (24)
0.364 (25) 0.365 (26) 0.365 (27) 0.366 (28) 0.365 (29)
Average = 0.365 Range = 0.003

rule 98
0.467 (0) 0.467 (1) 0.467 (2) 0.467 (3) 0.467 (4)
0.467 (5) 0.467 (6) 0.467 (7) 0.467 (8) 0.467 (9)
0.467 (10) 0.467 (11) 0.467 (12) 0.467 (13) 0.467 (14)
0.467 (15) 0.467 (16) 0.466 (17) 0.466 (18) 0.467 (19)
0.467 (20) 0.467 (21) 0.467 (22) 0.467 (23) 0.467 (24)
0.467 (25) 0.467 (26) 0.467 (27) 0.467 (28) 0.467 (29)
Average = 0.467 Range = 0.001

rule 99
0.533 (0) 0.533 (1) 0.533 (2) 0.533 (3) 0.533 (4)
0.533 (5) 0.533 (6) 0.533 (7) 0.533 (8) 0.533 (9)
0.533 (10) 0.533 (11) 0.533 (12) 0.533 (13) 0.533 (14)
0.533 (15) 0.533 (16) 0.533 (17) 0.533 (18) 0.533 (19)
0.533 (20) 0.533 (21) 0.533 (22) 0.533 (23) 0.533 (24)
0.533 (25) 0.533 (26) 0.533 (27) 0.533 (28) 0.533 (29)
Average = 0.533 Range = 0.000

rule 100
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 1.000 (4)
0.000 (5) 0.000 (6) 0.000 (7) 0.000 (8) 1.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 1.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 1.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
1.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 1.000 (29)
Average = 0.200 Range = 1.000

rule 101
0.498 (0) 0.505 (1) 0.492 (2) 0.508 (3) 0.490 (4)
0.498 (5) 0.500 (6) 0.508 (7) 0.491 (8) 0.507 (9)
0.498 (10) 0.501 (11) 0.502 (12) 0.497 (13) 0.501 (14)
0.503 (15) 0.499 (16) 0.506 (17) 0.495 (18) 0.497 (19)
0.504 (20) 0.509 (21) 0.495 (22) 0.509 (23) 0.493 (24)
0.509 (25) 0.501 (26) 0.496 (27) 0.508 (28) 0.498 (29)
Average = 0.501 Range = 0.019

rule 102
0.600 (0) 0.333 (1) 0.533 (2) 0.467 (3) 0.600 (4)
0.533 (5) 0.400 (6) 0.533 (7) 0.533 (8) 0.600 (9)
0.533 (10) 0.400 (11) 0.400 (12) 0.533 (13) 0.600 (14)
0.533 (15) 0.467 (16) 0.467 (17) 0.533 (18) 0.600 (19)
0.600 (20) 0.600 (21) 0.533 (22) 0.467 (23) 0.533 (24)
0.534 (25) 0.467 (26) 0.600 (27) 0.400 (28) 0.533 (29)
Average = 0.516 Range = 0.267

rule 103
0.550 (0) 0.550 (1) 0.550 (2) 0.550 (3) 0.550 (4)
0.550 (5) 0.550 (6) 0.550 (7) 0.550 (8) 0.550 (9)
0.550 (10) 0.550 (11) 0.550 (12) 0.550 (13) 0.550 (14)
0.550 (15) 0.550 (16) 0.550 (17) 0.550 (18) 0.550 (19)
0.550 (20) 0.550 (21) 0.550 (22) 0.550 (23) 0.550 (24)
0.550 (25) 0.550 (26) 0.550 (27) 0.550 (28) 0.550 (29)
Average = 0.550 Range = 0.000

rule 104
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 0.000 (4)
0.000 (5) 0.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
1.000 (10) 1.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
0.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 0.000 (29)
Average = 0.067 Range = 1.000

rule 105
0.500 (0) 0.567 (1) 0.433 (2) 0.500 (3) 0.433 (4)
0.500 (5) 0.500 (6) 0.433 (7) 0.567 (8) 0.433 (9)
0.500 (10) 0.567 (11) 0.500 (12) 0.433 (13) 0.633 (14)
0.567 (15) 0.500 (16) 0.700 (17) 0.367 (18) 0.500 (19)
0.566 (20) 0.500 (21) 0.500 (22) 0.567 (23) 0.433 (24)
0.767 (25) 0.433 (26) 0.500 (27) 0.367 (28) 0.500 (29)
Average = 0.509 Range = 0.400

rule 106
0.502 (0) 0.502 (1) 0.508 (2) 0.506 (3) 0.502 (4)
0.499 (5) 0.498 (6) 0.500 (7) 0.502 (8) 0.498 (9)
0.510 (10) 0.502 (11) 0.508 (12) 0.502 (13) 0.505 (14)
0.502 (15) 0.497 (16) 0.504 (17) 0.500 (18) 0.503 (19)
0.506 (20) 0.502 (21) 0.505 (22) 0.504 (23) 0.501 (24)
0.496 (25) 0.506 (26) 0.498 (27) 0.501 (28) 0.497 (29)
Average = 0.502 Range = 0.014

rule 107
0.635 (0) 0.635 (1) 0.636 (2) 0.635 (3) 0.635 (4)
0.635 (5) 0.635 (6) 0.635 (7) 0.635 (8) 0.635 (9)
0.635 (10) 0.635 (11) 0.635 (12) 0.636 (13) 0.635 (14)
0.635 (15) 0.635 (16) 0.635 (17) 0.635 (18) 0.635 (19)
0.636 (20) 0.635 (21) 0.635 (22) 0.635 (23) 0.636 (24)
0.635 (25) 0.635 (26) 0.635 (27) 0.636 (28) 0.635 (29)
Average = 0.635 Range = 0.001

rule 108
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 1.000 (4)
0.000 (5) 0.000 (6) 1.000 (7) 0.500 (8) 1.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
1.000 (15) 0.500 (16) 1.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 1.000 (21) 0.000 (22) 0.000 (23) 1.000 (24)
1.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 1.000 (29)
Average = 0.333 Range = 1.000

rule 109
0.000 (0) 1.000 (1) 0.500 (2) 1.000 (3) 0.000 (4)
0.000 (5) 1.000 (6) 0.500 (7) 1.000 (8) 0.000 (9)
0.000 (10) 1.000 (11) 0.500 (12) 1.000 (13) 0.000 (14)
0.001 (15) 1.000 (16) 0.500 (17) 1.000 (18) 0.500 (19)
0.999 (20) 0.500 (21) 1.000 (22) 0.000 (23) 0.000 (24)
1.000 (25) 0.000 (26) 0.000 (27) 1.000 (28) 0.000 (29)
Average = 0.500 Range = 1.000

rule 110
0.732 (0) 0.434 (1) 0.733 (2) 0.434 (3) 0.733 (4)
0.434 (5) 0.733 (6) 0.433 (7) 0.733 (8) 0.434 (9)
0.733 (10) 0.435 (11) 0.732 (12) 0.434 (13) 0.731 (14)
0.434 (15) 0.732 (16) 0.435 (17) 0.732 (18) 0.433 (19)
0.731 (20) 0.435 (21) 0.733 (22) 0.434 (23) 0.732 (24)
0.434 (25) 0.732 (26) 0.435 (27) 0.732 (28) 0.433 (29)
Average = 0.583 Range = 0.300

rule 111
0.400 (0) 0.800 (1) 0.400 (2) 0.799 (3) 0.400 (4)
0.799 (5) 0.401 (6) 0.799 (7) 0.401 (8) 0.799 (9)
0.401 (10) 0.799 (11) 0.401 (12) 0.799 (13) 0.401 (14)
0.799 (15) 0.401 (16) 0.801 (17) 0.400 (18) 0.800 (19)
0.400 (20) 0.800 (21) 0.400 (22) 0.800 (23) 0.400 (24)
0.800 (25) 0.400 (26) 0.800 (27) 0.400 (28) 0.799 (29)
Average = 0.600 Range = 0.401

rule 112
0.300 (0) 0.300 (1) 0.300 (2) 0.300 (3) 0.300 (4)
0.300 (5) 0.300 (6) 0.300 (7) 0.300 (8) 0.300 (9)
0.300 (10) 0.300 (11) 0.300 (12) 0.300 (13) 0.300 (14)
0.300 (15) 0.300 (16) 0.300 (17) 0.300 (18) 0.300 (19)
0.300 (20) 0.300 (21) 0.300 (22) 0.300 (23) 0.300 (24)
0.300 (25) 0.300 (26) 0.300 (27) 0.300 (28) 0.300 (29)
Average = 0.300 Range = 0.000

rule 113
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.500 (28) 0.500 (29)
Average = 0.500 Range = 0.000

rule 114
0.667 (0) 0.667 (1) 0.667 (2) 0.666 (3) 0.667 (4)
0.667 (5) 0.666 (6) 0.667 (7) 0.667 (8) 0.666 (9)
0.667 (10) 0.667 (11) 0.667 (12) 0.667 (13) 0.667 (14)
0.667 (15) 0.667 (16) 0.667 (17) 0.666 (18) 0.667 (19)
0.667 (20) 0.667 (21) 0.667 (22) 0.667 (23) 0.667 (24)
0.667 (25) 0.667 (26) 0.667 (27) 0.667 (28) 0.667 (29)
Average = 0.667 Range = 0.000

rule 115
0.650 (0) 0.650 (1) 0.650 (2) 0.650 (3) 0.650 (4)
0.650 (5) 0.650 (6) 0.650 (7) 0.650 (8) 0.650 (9)
0.650 (10) 0.650 (11) 0.650 (12) 0.650 (13) 0.650 (14)
0.650 (15) 0.650 (16) 0.650 (17) 0.650 (18) 0.650 (19)
0.650 (20) 0.650 (21) 0.650 (22) 0.650 (23) 0.650 (24)
0.650 (25) 0.650 (26) 0.650 (27) 0.650 (28) 0.650 (29)
Average = 0.650 Range = 0.000

rule 116
0.467 (0) 0.467 (1) 0.467 (2) 0.467 (3) 0.467 (4)
0.467 (5) 0.467 (6) 0.467 (7) 0.467 (8) 0.467 (9)
0.467 (10) 0.467 (11) 0.467 (12) 0.467 (13) 0.467 (14)
0.467 (15) 0.466 (16) 0.466 (17) 0.467 (18) 0.467 (19)
0.467 (20) 0.467 (21) 0.467 (22) 0.467 (23) 0.467 (24)
0.467 (25) 0.467 (26) 0.467 (27) 0.467 (28) 0.467 (29)
Average = 0.467 Range = 0.001

rule 117
0.467 (0) 0.533 (1) 0.467 (2) 0.533 (3) 0.467 (4)
0.533 (5) 0.467 (6) 0.533 (7) 0.467 (8) 0.533 (9)
0.467 (10) 0.533 (11) 0.467 (12) 0.533 (13) 0.467 (14)
0.533 (15) 0.467 (16) 0.533 (17) 0.467 (18) 0.533 (19)
0.467 (20) 0.533 (21) 0.467 (22) 0.533 (23) 0.467 (24)
0.533 (25) 0.467 (26) 0.533 (27) 0.467 (28) 0.533 (29)
Average = 0.500 Range = 0.067

rule 118
0.335 (0) 0.335 (1) 0.666 (2) 0.666 (3) 0.666 (4)
0.666 (5) 0.666 (6) 0.666 (7) 0.666 (8) 0.666 (9)
0.667 (10) 0.666 (11) 0.666 (12) 0.667 (13) 0.667 (14)
0.666 (15) 0.667 (16) 0.667 (17) 0.666 (18) 0.667 (19)
0.667 (20) 0.667 (21) 0.667 (22) 0.667 (23) 0.667 (24)
0.667 (25) 0.667 (26) 0.665 (27) 0.666 (28) 0.666 (29)
Average = 0.644 Range = 0.332

rule 119
0.533 (0) 0.533 (1) 0.533 (2) 0.533 (3) 0.533 (4)
0.533 (5) 0.533 (6) 0.534 (7) 0.533 (8) 0.534 (9)
0.533 (10) 0.533 (11) 0.533 (12) 0.533 (13) 0.533 (14)
0.534 (15) 0.533 (16) 0.533 (17) 0.533 (18) 0.533 (19)
0.533 (20) 0.533 (21) 0.533 (22) 0.533 (23) 0.533 (24)
0.533 (25) 0.533 (26) 0.533 (27) 0.533 (28) 0.533 (29)
Average = 0.533 Range = 0.000

rule 120
0.496 (0) 0.500 (1) 0.495 (2) 0.498 (3) 0.491 (4)
0.497 (5) 0.492 (6) 0.501 (7) 0.496 (8) 0.502 (9)
0.494 (10) 0.493 (11) 0.490 (12) 0.503 (13) 0.495 (14)
0.500 (15) 0.504 (16) 0.507 (17) 0.499 (18) 0.500 (19)
0.497 (20) 0.502 (21) 0.492 (22) 0.496 (23) 0.497 (24)
0.496 (25) 0.490 (26) 0.486 (27) 0.486 (28) 0.489 (29)
Average = 0.496 Range = 0.021

rule 121
0.500 (0) 0.700 (1) 0.500 (2) 0.700 (3) 0.501 (4)
0.700 (5) 0.501 (6) 0.699 (7) 0.501 (8) 0.700 (9)
0.501 (10) 0.699 (11) 0.501 (12) 0.700 (13) 0.501 (14)
0.700 (15) 0.500 (16) 0.700 (17) 0.500 (18) 0.700 (19)
0.500 (20) 0.700 (21) 0.500 (22) 0.700 (23) 0.500 (24)
0.700 (25) 0.500 (26) 0.701 (27) 0.500 (28) 0.700 (29)
Average = 0.600 Range = 0.201

rule 122
0.002 (0) 0.002 (1) 0.002 (2) 0.002 (3) 0.002 (4)
0.002 (5) 0.002 (6) 0.002 (7) 0.002 (8) 0.002 (9)
0.002 (10) 0.002 (11) 0.002 (12) 0.002 (13) 0.002 (14)
0.002 (15) 0.003 (16) 0.002 (17) 0.002 (18) 0.002 (19)
0.002 (20) 0.002 (21) 0.002 (22) 0.002 (23) 0.002 (24)
0.002 (25) 0.002 (26) 0.002 (27) 0.002 (28) 0.002 (29)
Average = 0.002 Range = 0.001

rule 123
1.000 (0) 0.500 (1) 0.500 (2) 1.000 (3) 0.500 (4)
0.500 (5) 0.500 (6) 1.000 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 1.000 (14)
0.500 (15) 0.500 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 1.000 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 1.000 (28) 0.500 (29)
Average = 0.600 Range = 0.500

rule 124
0.433 (0) 0.733 (1) 0.433 (2) 0.733 (3) 0.433 (4)
0.732 (5) 0.433 (6) 0.732 (7) 0.434 (8) 0.732 (9)
0.434 (10) 0.732 (11) 0.433 (12) 0.732 (13) 0.433 (14)
0.734 (15) 0.434 (16) 0.733 (17) 0.434 (18) 0.733 (19)
0.433 (20) 0.733 (21) 0.433 (22) 0.733 (23) 0.433 (24)
0.734 (25) 0.433 (26) 0.733 (27) 0.433 (28) 0.734 (29)
Average = 0.583 Range = 0.301

rule 125
0.633 (0) 0.500 (1) 0.633 (2) 0.500 (3) 0.633 (4)
0.500 (5) 0.633 (6) 0.500 (7) 0.633 (8) 0.500 (9)
0.634 (10) 0.500 (11) 0.634 (12) 0.500 (13) 0.634 (14)
0.500 (15) 0.633 (16) 0.500 (17) 0.633 (18) 0.500 (19)
0.633 (20) 0.500 (21) 0.633 (22) 0.500 (23) 0.633 (24)
0.500 (25) 0.633 (26) 0.500 (27) 0.633 (28) 0.500 (29)
Average = 0.567 Range = 0.134

rule 126
0.001 (0) 0.001 (1) 0.001 (2) 0.001 (3) 0.001 (4)
0.002 (5) 0.002 (6) 0.001 (7) 0.001 (8) 0.002 (9)
0.001 (10) 0.001 (11) 0.002 (12) 0.001 (13) 0.001 (14)
0.001 (15) 0.001 (16) 0.001 (17) 0.001 (18) 0.001 (19)
0.002 (20) 0.001 (21) 0.001 (22) 0.001 (23) 0.001 (24)
0.001 (25) 0.002 (26) 0.001 (27) 0.001 (28) 0.001 (29)
Average = 0.001 Range = 0.001

rule 127
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 1.000 (17) 0.500 (18) 0.500 (19)
0.500 (20) 1.000 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.500 (28) 0.500 (29)
Average = 0.533 Range = 0.500

rule 128
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 0.000 (4)
0.000 (5) 0.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
0.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 0.000 (29)
Average = 0.000 Range = 0.000

rule 129
0.999 (0) 0.998 (1) 0.999 (2) 0.999 (3) 0.999 (4)
0.999 (5) 0.998 (6) 0.999 (7) 0.999 (8) 0.998 (9)
0.998 (10) 0.999 (11) 0.998 (12) 0.998 (13) 0.998 (14)
0.998 (15) 0.998 (16) 0.998 (17) 0.998 (18) 0.998 (19)
0.998 (20) 0.998 (21) 0.999 (22) 0.999 (23) 0.998 (24)
0.999 (25) 0.998 (26) 0.998 (27) 0.998 (28) 0.999 (29)
Average = 0.999 Range = 0.001

rule 130
0.133 (0) 0.133 (1) 0.133 (2) 0.133 (3) 0.133 (4)
0.133 (5) 0.133 (6) 0.133 (7) 0.133 (8) 0.133 (9)
0.133 (10) 0.133 (11) 0.133 (12) 0.133 (13) 0.133 (14)
0.133 (15) 0.133 (16) 0.133 (17) 0.133 (18) 0.133 (19)
0.133 (20) 0.133 (21) 0.133 (22) 0.133 (23) 0.133 (24)
0.133 (25) 0.133 (26) 0.133 (27) 0.133 (28) 0.133 (29)
Average = 0.133 Range = 0.000

rule 131
0.333 (0) 0.334 (1) 0.334 (2) 0.333 (3) 0.333 (4)
0.333 (5) 0.333 (6) 0.333 (7) 0.666 (8) 0.667 (9)
0.333 (10) 0.333 (11) 0.667 (12) 0.667 (13) 0.333 (14)
0.333 (15) 0.333 (16) 0.333 (17) 0.667 (18) 0.667 (19)
0.333 (20) 0.333 (21) 0.333 (22) 0.333 (23) 0.333 (24)
0.333 (25) 0.333 (26) 0.333 (27) 0.333 (28) 0.333 (29)
Average = 0.400 Range = 0.333

rule 132
0.000 (0) 0.000 (1) 1.000 (2) 0.000 (3) 0.000 (4)
0.000 (5) 1.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
0.000 (10) 1.000 (11) 0.000 (12) 1.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
1.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 0.000 (29)
Average = 0.167 Range = 1.000

rule 133
0.000 (0) 0.000 (1) 1.000 (2) 0.000 (3) 1.000 (4)
0.000 (5) 0.500 (6) 0.500 (7) 0.000 (8) 1.000 (9)
0.000 (10) 1.000 (11) 0.000 (12) 1.000 (13) 0.000 (14)
1.000 (15) 0.000 (16) 0.000 (17) 1.000 (18) 0.000 (19)
0.000 (20) 1.000 (21) 0.000 (22) 1.000 (23) 0.000 (24)
0.000 (25) 1.000 (26) 0.000 (27) 0.000 (28) 1.000 (29)
Average = 0.400 Range = 1.000

rule 134
0.200 (0) 0.400 (1) 0.200 (2) 0.400 (3) 0.200 (4)
0.400 (5) 0.200 (6) 0.400 (7) 0.200 (8) 0.400 (9)
0.200 (10) 0.400 (11) 0.200 (12) 0.400 (13) 0.200 (14)
0.400 (15) 0.200 (16) 0.400 (17) 0.200 (18) 0.400 (19)
0.200 (20) 0.400 (21) 0.200 (22) 0.400 (23) 0.200 (24)
0.400 (25) 0.200 (26) 0.400 (27) 0.200 (28) 0.400 (29)
Average = 0.300 Range = 0.200

rule 135
0.497 (0) 0.501 (1) 0.503 (2) 0.502 (3) 0.501 (4)
0.491 (5) 0.509 (6) 0.507 (7) 0.488 (8) 0.500 (9)
0.494 (10) 0.502 (11) 0.498 (12) 0.499 (13) 0.502 (14)
0.504 (15) 0.491 (16) 0.501 (17) 0.501 (18) 0.495 (19)
0.507 (20) 0.495 (21) 0.495 (22) 0.505 (23) 0.502 (24)
0.506 (25) 0.494 (26) 0.507 (27) 0.504 (28) 0.492 (29)
Average = 0.500 Range = 0.021

rule 136
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 0.000 (4)
0.000 (5) 0.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
0.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 0.000 (29)
Average = 0.000 Range = 0.000

rule 137
0.423 (0) 0.422 (1) 0.422 (2) 0.423 (3) 0.423 (4)
0.423 (5) 0.423 (6) 0.423 (7) 0.423 (8) 0.422 (9)
0.423 (10) 0.422 (11) 0.423 (12) 0.422 (13) 0.423 (14)
0.422 (15) 0.423 (16) 0.422 (17) 0.423 (18) 0.423 (19)
0.423 (20) 0.423 (21) 0.423 (22) 0.422 (23) 0.422 (24)
0.423 (25) 0.423 (26) 0.423 (27) 0.423 (28) 0.422 (29)
Average = 0.423 Range = 0.001

rule 138
0.333 (0) 0.333 (1) 0.333 (2) 0.333 (3) 0.333 (4)
0.333 (5) 0.333 (6) 0.333 (7) 0.333 (8) 0.333 (9)
0.333 (10) 0.333 (11) 0.333 (12) 0.333 (13) 0.333 (14)
0.334 (15) 0.334 (16) 0.334 (17) 0.334 (18) 0.333 (19)
0.333 (20) 0.333 (21) 0.333 (22) 0.333 (23) 0.333 (24)
0.333 (25) 0.333 (26) 0.333 (27) 0.333 (28) 0.333 (29)
Average = 0.333 Range = 0.001

rule 139
0.533 (0) 0.533 (1) 0.533 (2) 0.533 (3) 0.533 (4)
0.533 (5) 0.533 (6) 0.533 (7) 0.533 (8) 0.533 (9)
0.533 (10) 0.533 (11) 0.533 (12) 0.533 (13) 0.533 (14)
0.533 (15) 0.533 (16) 0.533 (17) 0.534 (18) 0.533 (19)
0.533 (20) 0.533 (21) 0.533 (22) 0.533 (23) 0.533 (24)
0.533 (25) 0.533 (26) 0.533 (27) 0.533 (28) 0.533 (29)
Average = 0.533 Range = 0.000

rule 140
1.000 (0) 0.000 (1) 0.000 (2) 1.000 (3) 0.000 (4)
1.000 (5) 0.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
1.000 (10) 0.000 (11) 1.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 1.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
1.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
1.000 (25) 0.000 (26) 0.000 (27) 1.000 (28) 0.000 (29)
Average = 0.300 Range = 1.000

rule 141
0.000 (0) 1.000 (1) 0.000 (2) 0.000 (3) 1.000 (4)
0.000 (5) 0.000 (6) 1.000 (7) 0.000 (8) 1.000 (9)
0.000 (10) 1.000 (11) 0.000 (12) 1.000 (13) 0.000 (14)
1.000 (15) 0.000 (16) 1.000 (17) 0.000 (18) 0.000 (19)
1.000 (20) 0.000 (21) 1.000 (22) 0.000 (23) 1.000 (24)
0.000 (25) 0.000 (26) 1.000 (27) 0.000 (28) 1.000 (29)
Average = 0.433 Range = 1.000

rule 142
0.467 (0) 0.533 (1) 0.467 (2) 0.533 (3) 0.467 (4)
0.533 (5) 0.467 (6) 0.533 (7) 0.467 (8) 0.533 (9)
0.467 (10) 0.533 (11) 0.467 (12) 0.533 (13) 0.467 (14)
0.533 (15) 0.467 (16) 0.533 (17) 0.467 (18) 0.533 (19)
0.467 (20) 0.533 (21) 0.467 (22) 0.533 (23) 0.467 (24)
0.533 (25) 0.467 (26) 0.534 (27) 0.467 (28) 0.533 (29)
Average = 0.500 Range = 0.067

rule 143
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.500 (28) 0.500 (29)
Average = 0.500 Range = 0.001

rule 144
0.133 (0) 0.133 (1) 0.133 (2) 0.133 (3) 0.133 (4)
0.133 (5) 0.133 (6) 0.133 (7) 0.133 (8) 0.133 (9)
0.133 (10) 0.133 (11) 0.133 (12) 0.133 (13) 0.133 (14)
0.133 (15) 0.133 (16) 0.133 (17) 0.133 (18) 0.133 (19)
0.133 (20) 0.134 (21) 0.133 (22) 0.133 (23) 0.133 (24)
0.133 (25) 0.133 (26) 0.133 (27) 0.133 (28) 0.133 (29)
Average = 0.133 Range = 0.000

rule 145
0.333 (0) 0.667 (1) 0.333 (2) 0.333 (3) 0.333 (4)
0.333 (5) 0.333 (6) 0.667 (7) 0.333 (8) 0.333 (9)
0.333 (10) 0.667 (11) 0.667 (12) 0.333 (13) 0.333 (14)
0.334 (15) 0.333 (16) 0.333 (17) 0.666 (18) 0.334 (19)
0.334 (20) 0.333 (21) 0.333 (22) 0.333 (23) 0.333 (24)
0.333 (25) 0.333 (26) 0.333 (27) 0.333 (28) 0.333 (29)
Average = 0.389 Range = 0.333

rule 146
0.001 (0) 0.001 (1) 0.001 (2) 0.001 (3) 0.001 (4)
0.001 (5) 0.001 (6) 0.001 (7) 0.000 (8) 0.001 (9)
0.000 (10) 0.001 (11) 0.000 (12) 0.001 (13) 0.001 (14)
0.001 (15) 0.001 (16) 0.001 (17) 0.001 (18) 0.001 (19)
0.000 (20) 0.001 (21) 0.001 (22) 0.000 (23) 0.001 (24)
0.001 (25) 0.000 (26) 0.001 (27) 0.001 (28) 0.001 (29)
Average = 0.001 Range = 0.001

rule 147
0.531 (0) 0.530 (1) 0.532 (2) 0.530 (3) 0.532 (4)
0.530 (5) 0.532 (6) 0.531 (7) 0.532 (8) 0.530 (9)
0.532 (10) 0.531 (11) 0.532 (12) 0.531 (13) 0.532 (14)
0.531 (15) 0.532 (16) 0.531 (17) 0.533 (18) 0.531 (19)
0.533 (20) 0.531 (21) 0.532 (22) 0.531 (23) 0.532 (24)
0.531 (25) 0.532 (26) 0.531 (27) 0.532 (28) 0.530 (29)
Average = 0.531 Range = 0.002

rule 148
0.200 (0) 0.300 (1) 0.200 (2) 0.300 (3) 0.200 (4)
0.300 (5) 0.200 (6) 0.300 (7) 0.200 (8) 0.300 (9)
0.200 (10) 0.300 (11) 0.200 (12) 0.300 (13) 0.200 (14)
0.300 (15) 0.200 (16) 0.300 (17) 0.200 (18) 0.300 (19)
0.200 (20) 0.300 (21) 0.200 (22) 0.300 (23) 0.200 (24)
0.300 (25) 0.200 (26) 0.300 (27) 0.200 (28) 0.300 (29)
Average = 0.250 Range = 0.100

rule 149
0.499 (0) 0.504 (1) 0.494 (2) 0.505 (3) 0.497 (4)
0.504 (5) 0.495 (6) 0.500 (7) 0.496 (8) 0.498 (9)
0.497 (10) 0.502 (11) 0.503 (12) 0.500 (13) 0.500 (14)
0.496 (15) 0.499 (16) 0.495 (17) 0.506 (18) 0.498 (19)
0.498 (20) 0.506 (21) 0.502 (22) 0.494 (23) 0.498 (24)
0.501 (25) 0.499 (26) 0.500 (27) 0.499 (28) 0.503 (29)
Average = 0.500 Range = 0.012

rule 150
0.400 (0) 0.400 (1) 0.667 (2) 0.534 (3) 0.533 (4)
0.533 (5) 0.533 (6) 0.533 (7) 0.667 (8) 0.533 (9)
0.533 (10) 0.734 (11) 0.533 (12) 0.467 (13) 0.533 (14)
0.533 (15) 0.400 (16) 0.467 (17) 0.400 (18) 0.733 (19)
0.533 (20) 0.333 (21) 0.666 (22) 0.533 (23) 0.533 (24)
0.466 (25) 0.400 (26) 0.467 (27) 0.667 (28) 0.533 (29)
Average = 0.527 Range = 0.400

rule 151
0.999 (0) 0.999 (1) 0.999 (2) 0.999 (3) 0.999 (4)
0.999 (5) 0.999 (6) 0.999 (7) 0.999 (8) 0.999 (9)
0.999 (10) 0.998 (11) 0.998 (12) 0.999 (13) 0.999 (14)
0.998 (15) 0.998 (16) 0.999 (17) 0.999 (18) 0.999 (19)
0.999 (20) 0.999 (21) 0.999 (22) 0.999 (23) 0.999 (24)
0.999 (25) 0.999 (26) 0.998 (27) 0.999 (28) 0.999 (29)
Average = 0.999 Range = 0.001

rule 152
0.167 (0) 0.167 (1) 0.167 (2) 0.167 (3) 0.167 (4)
0.167 (5) 0.167 (6) 0.167 (7) 0.167 (8) 0.167 (9)
0.167 (10) 0.167 (11) 0.167 (12) 0.167 (13) 0.167 (14)
0.167 (15) 0.167 (16) 0.167 (17) 0.167 (18) 0.167 (19)
0.167 (20) 0.167 (21) 0.167 (22) 0.167 (23) 0.167 (24)
0.167 (25) 0.167 (26) 0.167 (27) 0.167 (28) 0.167 (29)
Average = 0.167 Range = 0.000

rule 153
0.533 (0) 0.533 (1) 0.400 (2) 0.467 (3) 0.400 (4)
0.467 (5) 0.667 (6) 0.534 (7) 0.467 (8) 0.467 (9)
0.600 (10) 0.467 (11) 0.600 (12) 0.400 (13) 0.533 (14)
0.467 (15) 0.466 (16) 0.467 (17) 0.534 (18) 0.400 (19)
0.400 (20) 0.600 (21) 0.533 (22) 0.467 (23) 0.467 (24)
0.400 (25) 0.533 (26) 0.600 (27) 0.533 (28) 0.467 (29)
Average = 0.496 Range = 0.267

rule 154
0.567 (0) 0.533 (1) 0.567 (2) 0.533 (3) 0.567 (4)
0.533 (5) 0.567 (6) 0.533 (7) 0.567 (8) 0.533 (9)
0.567 (10) 0.533 (11) 0.567 (12) 0.533 (13) 0.567 (14)
0.533 (15) 0.567 (16) 0.533 (17) 0.567 (18) 0.533 (19)
0.567 (20) 0.533 (21) 0.567 (22) 0.534 (23) 0.567 (24)
0.533 (25) 0.567 (26) 0.533 (27) 0.567 (28) 0.533 (29)
Average = 0.550 Range = 0.034

rule 155
0.667 (0) 0.633 (1) 0.666 (2) 0.633 (3) 0.666 (4)
0.633 (5) 0.667 (6) 0.633 (7) 0.667 (8) 0.633 (9)
0.667 (10) 0.633 (11) 0.667 (12) 0.633 (13) 0.667 (14)
0.633 (15) 0.667 (16) 0.633 (17) 0.667 (18) 0.633 (19)
0.667 (20) 0.633 (21) 0.667 (22) 0.633 (23) 0.667 (24)
0.633 (25) 0.667 (26) 0.633 (27) 0.667 (28) 0.633 (29)
Average = 0.650 Range = 0.034

rule 156
0.000 (0) 1.000 (1) 0.500 (2) 0.000 (3) 1.000 (4)
0.500 (5) 0.000 (6) 1.000 (7) 0.000 (8) 1.000 (9)
0.000 (10) 1.000 (11) 0.000 (12) 1.000 (13) 0.000 (14)
1.000 (15) 0.000 (16) 1.000 (17) 0.500 (18) 0.000 (19)
1.000 (20) 0.000 (21) 1.000 (22) 0.000 (23) 1.000 (24)
0.000 (25) 1.000 (26) 0.500 (27) 0.000 (28) 1.000 (29)
Average = 0.500 Range = 1.000

rule 157
0.000 (0) 1.000 (1) 0.500 (2) 0.000 (3) 1.000 (4)
0.000 (5) 1.000 (6) 0.000 (7) 1.000 (8) 0.500 (9)
0.000 (10) 1.000 (11) 0.000 (12) 1.000 (13) 0.500 (14)
0.000 (15) 1.000 (16) 0.000 (17) 1.000 (18) 0.500 (19)
0.000 (20) 1.000 (21) 0.000 (22) 1.000 (23) 0.000 (24)
1.000 (25) 0.000 (26) 1.000 (27) 0.000 (28) 1.000 (29)
Average = 0.500 Range = 1.000

rule 158
0.700 (0) 0.700 (1) 0.700 (2) 0.700 (3) 0.700 (4)
0.700 (5) 0.700 (6) 0.700 (7) 0.700 (8) 0.700 (9)
0.700 (10) 0.700 (11) 0.700 (12) 0.700 (13) 0.700 (14)
0.700 (15) 0.700 (16) 0.700 (17) 0.700 (18) 0.700 (19)
0.700 (20) 0.700 (21) 0.700 (22) 0.700 (23) 0.700 (24)
0.700 (25) 0.700 (26) 0.700 (27) 0.700 (28) 0.700 (29)
Average = 0.700 Range = 0.001

rule 159
0.800 (0) 0.800 (1) 0.800 (2) 0.800 (3) 0.800 (4)
0.800 (5) 0.800 (6) 0.800 (7) 0.800 (8) 0.800 (9)
0.800 (10) 0.800 (11) 0.800 (12) 0.800 (13) 0.800 (14)
0.800 (15) 0.800 (16) 0.800 (17) 0.800 (18) 0.800 (19)
0.800 (20) 0.800 (21) 0.800 (22) 0.800 (23) 0.800 (24)
0.800 (25) 0.800 (26) 0.800 (27) 0.800 (28) 0.800 (29)
Average = 0.800 Range = 0.000

rule 160
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 0.000 (4)
0.000 (5) 0.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
0.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 0.000 (29)
Average = 0.000 Range = 0.000

rule 161
0.998 (0) 0.997 (1) 0.998 (2) 0.998 (3) 0.998 (4)
0.998 (5) 0.998 (6) 0.998 (7) 0.998 (8) 0.998 (9)
0.998 (10) 0.998 (11) 0.998 (12) 0.997 (13) 0.998 (14)
0.998 (15) 0.997 (16) 0.998 (17) 0.998 (18) 0.998 (19)
0.998 (20) 0.998 (21) 0.998 (22) 0.998 (23) 0.998 (24)
0.998 (25) 0.998 (26) 0.998 (27) 0.997 (28) 0.998 (29)
Average = 0.998 Range = 0.001

rule 162
0.367 (0) 0.367 (1) 0.367 (2) 0.367 (3) 0.367 (4)
0.367 (5) 0.367 (6) 0.367 (7) 0.367 (8) 0.367 (9)
0.367 (10) 0.367 (11) 0.367 (12) 0.367 (13) 0.367 (14)
0.367 (15) 0.367 (16) 0.367 (17) 0.367 (18) 0.367 (19)
0.367 (20) 0.367 (21) 0.367 (22) 0.367 (23) 0.367 (24)
0.367 (25) 0.367 (26) 0.367 (27) 0.367 (28) 0.367 (29)
Average = 0.367 Range = 0.000

rule 163
0.367 (0) 0.367 (1) 0.367 (2) 0.367 (3) 0.367 (4)
0.367 (5) 0.367 (6) 0.367 (7) 0.367 (8) 0.367 (9)
0.367 (10) 0.367 (11) 0.367 (12) 0.367 (13) 0.367 (14)
0.367 (15) 0.367 (16) 0.367 (17) 0.367 (18) 0.367 (19)
0.367 (20) 0.367 (21) 0.367 (22) 0.367 (23) 0.367 (24)
0.367 (25) 0.367 (26) 0.367 (27) 0.367 (28) 0.367 (29)
Average = 0.367 Range = 0.000

rule 164
0.000 (0) 0.000 (1) 1.000 (2) 0.000 (3) 0.000 (4)
0.000 (5) 0.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
1.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
0.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 0.000 (29)
Average = 0.067 Range = 1.000

rule 165
0.400 (0) 0.567 (1) 0.567 (2) 0.667 (3) 0.500 (4)
0.500 (5) 0.467 (6) 0.433 (7) 0.500 (8) 0.400 (9)
0.500 (10) 0.567 (11) 0.467 (12) 0.500 (13) 0.367 (14)
0.400 (15) 0.567 (16) 0.567 (17) 0.666 (18) 0.500 (19)
0.500 (20) 0.467 (21) 0.433 (22) 0.500 (23) 0.400 (24)
0.500 (25) 0.567 (26) 0.467 (27) 0.500 (28) 0.367 (29)
Average = 0.493 Range = 0.300

rule 166
0.550 (0) 0.534 (1) 0.550 (2) 0.533 (3) 0.550 (4)
0.533 (5) 0.550 (6) 0.533 (7) 0.550 (8) 0.533 (9)
0.550 (10) 0.533 (11) 0.550 (12) 0.534 (13) 0.550 (14)
0.534 (15) 0.550 (16) 0.533 (17) 0.550 (18) 0.533 (19)
0.550 (20) 0.533 (21) 0.550 (22) 0.533 (23) 0.550 (24)
0.533 (25) 0.550 (26) 0.534 (27) 0.550 (28) 0.533 (29)
Average = 0.542 Range = 0.017

rule 167
0.600 (0) 0.600 (1) 0.600 (2) 0.600 (3) 0.600 (4)
0.600 (5) 0.600 (6) 0.600 (7) 0.600 (8) 0.600 (9)
0.600 (10) 0.600 (11) 0.600 (12) 0.600 (13) 0.600 (14)
0.600 (15) 0.600 (16) 0.600 (17) 0.600 (18) 0.600 (19)
0.600 (20) 0.600 (21) 0.600 (22) 0.600 (23) 0.600 (24)
0.600 (25) 0.600 (26) 0.600 (27) 0.600 (28) 0.600 (29)
Average = 0.600 Range = 0.000

rule 168
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 0.000 (4)
0.000 (5) 0.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
0.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 0.000 (29)
Average = 0.000 Range = 0.000

rule 169
0.503 (0) 0.501 (1) 0.509 (2) 0.502 (3) 0.505 (4)
0.502 (5) 0.499 (6) 0.502 (7) 0.503 (8) 0.495 (9)
0.496 (10) 0.495 (11) 0.497 (12) 0.494 (13) 0.493 (14)
0.491 (15) 0.493 (16) 0.490 (17) 0.497 (18) 0.494 (19)
0.497 (20) 0.491 (21) 0.492 (22) 0.495 (23) 0.495 (24)
0.491 (25) 0.492 (26) 0.496 (27) 0.495 (28) 0.497 (29)
Average = 0.497 Range = 0.018

rule 170
0.567 (0) 0.567 (1) 0.567 (2) 0.567 (3) 0.567 (4)
0.567 (5) 0.567 (6) 0.567 (7) 0.567 (8) 0.567 (9)
0.567 (10) 0.567 (11) 0.567 (12) 0.567 (13) 0.567 (14)
0.567 (15) 0.567 (16) 0.567 (17) 0.567 (18) 0.567 (19)
0.567 (20) 0.567 (21) 0.567 (22) 0.567 (23) 0.567 (24)
0.567 (25) 0.567 (26) 0.567 (27) 0.567 (28) 0.567 (29)
Average = 0.567 Range = 0.000

rule 171
0.600 (0) 0.600 (1) 0.600 (2) 0.600 (3) 0.600 (4)
0.600 (5) 0.600 (6) 0.600 (7) 0.600 (8) 0.600 (9)
0.600 (10) 0.600 (11) 0.600 (12) 0.600 (13) 0.600 (14)
0.600 (15) 0.600 (16) 0.600 (17) 0.600 (18) 0.600 (19)
0.600 (20) 0.600 (21) 0.600 (22) 0.600 (23) 0.600 (24)
0.600 (25) 0.600 (26) 0.600 (27) 0.600 (28) 0.600 (29)
Average = 0.600 Range = 0.000

rule 172
0.000 (0) 1.000 (1) 0.000 (2) 0.000 (3) 0.000 (4)
0.000 (5) 1.000 (6) 0.001 (7) 0.001 (8) 0.001 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 1.000 (18) 0.000 (19)
0.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
1.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 0.000 (29)
Average = 0.133 Range = 1.000

rule 173
0.733 (0) 0.600 (1) 0.733 (2) 0.600 (3) 0.733 (4)
0.600 (5) 0.733 (6) 0.600 (7) 0.733 (8) 0.600 (9)
0.733 (10) 0.600 (11) 0.733 (12) 0.600 (13) 0.733 (14)
0.600 (15) 0.733 (16) 0.600 (17) 0.733 (18) 0.600 (19)
0.733 (20) 0.600 (21) 0.733 (22) 0.600 (23) 0.734 (24)
0.600 (25) 0.733 (26) 0.600 (27) 0.733 (28) 0.600 (29)
Average = 0.667 Range = 0.134

rule 174
0.667 (0) 0.667 (1) 0.667 (2) 0.667 (3) 0.667 (4)
0.667 (5) 0.667 (6) 0.667 (7) 0.667 (8) 0.667 (9)
0.667 (10) 0.667 (11) 0.667 (12) 0.667 (13) 0.667 (14)
0.666 (15) 0.667 (16) 0.667 (17) 0.666 (18) 0.667 (19)
0.667 (20) 0.667 (21) 0.667 (22) 0.667 (23) 0.667 (24)
0.667 (25) 0.667 (26) 0.667 (27) 0.667 (28) 0.667 (29)
Average = 0.667 Range = 0.000

rule 175
0.767 (0) 0.767 (1) 0.767 (2) 0.767 (3) 0.767 (4)
0.767 (5) 0.767 (6) 0.767 (7) 0.767 (8) 0.767 (9)
0.767 (10) 0.767 (11) 0.767 (12) 0.767 (13) 0.767 (14)
0.767 (15) 0.767 (16) 0.767 (17) 0.766 (18) 0.766 (19)
0.766 (20) 0.766 (21) 0.766 (22) 0.766 (23) 0.767 (24)
0.767 (25) 0.767 (26) 0.767 (27) 0.767 (28) 0.767 (29)
Average = 0.767 Range = 0.000

rule 176
0.333 (0) 0.333 (1) 0.333 (2) 0.333 (3) 0.333 (4)
0.333 (5) 0.333 (6) 0.333 (7) 0.333 (8) 0.333 (9)
0.333 (10) 0.333 (11) 0.333 (12) 0.333 (13) 0.333 (14)
0.333 (15) 0.333 (16) 0.333 (17) 0.333 (18) 0.333 (19)
0.333 (20) 0.333 (21) 0.333 (22) 0.333 (23) 0.333 (24)
0.333 (25) 0.333 (26) 0.333 (27) 0.333 (28) 0.333 (29)
Average = 0.333 Range = 0.000

rule 177
0.367 (0) 0.367 (1) 0.367 (2) 0.367 (3) 0.367 (4)
0.367 (5) 0.367 (6) 0.367 (7) 0.367 (8) 0.367 (9)
0.367 (10) 0.367 (11) 0.367 (12) 0.367 (13) 0.367 (14)
0.367 (15) 0.367 (16) 0.367 (17) 0.367 (18) 0.367 (19)
0.367 (20) 0.367 (21) 0.367 (22) 0.367 (23) 0.367 (24)
0.367 (25) 0.367 (26) 0.367 (27) 0.367 (28) 0.367 (29)
Average = 0.367 Range = 0.000

rule 178
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.500 (28) 0.500 (29)
Average = 0.500 Range = 0.000

rule 179
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.500 (28) 0.500 (29)
Average = 0.500 Range = 0.000

rule 180
0.567 (0) 0.567 (1) 0.567 (2) 0.567 (3) 0.567 (4)
0.567 (5) 0.567 (6) 0.567 (7) 0.567 (8) 0.567 (9)
0.567 (10) 0.567 (11) 0.567 (12) 0.567 (13) 0.567 (14)
0.567 (15) 0.567 (16) 0.567 (17) 0.567 (18) 0.567 (19)
0.567 (20) 0.567 (21) 0.567 (22) 0.567 (23) 0.566 (24)
0.566 (25) 0.567 (26) 0.567 (27) 0.567 (28) 0.567 (29)
Average = 0.567 Range = 0.001

rule 181
0.617 (0) 0.633 (1) 0.617 (2) 0.633 (3) 0.617 (4)
0.633 (5) 0.617 (6) 0.633 (7) 0.617 (8) 0.633 (9)
0.617 (10) 0.633 (11) 0.617 (12) 0.634 (13) 0.617 (14)
0.633 (15) 0.617 (16) 0.633 (17) 0.617 (18) 0.633 (19)
0.617 (20) 0.634 (21) 0.617 (22) 0.633 (23) 0.616 (24)
0.633 (25) 0.617 (26) 0.633 (27) 0.617 (28) 0.633 (29)
Average = 0.625 Range = 0.017

rule 182
0.999 (0) 0.999 (1) 0.999 (2) 0.999 (3) 0.999 (4)
0.999 (5) 0.999 (6) 0.999 (7) 0.999 (8) 0.999 (9)
0.999 (10) 0.999 (11) 0.999 (12) 0.999 (13) 0.999 (14)
0.999 (15) 0.999 (16) 0.999 (17) 0.999 (18) 0.999 (19)
0.999 (20) 0.999 (21) 0.999 (22) 0.999 (23) 0.999 (24)
0.999 (25) 0.999 (26) 0.999 (27) 0.999 (28) 1.000 (29)
Average = 0.999 Range = 0.001

rule 183
0.999 (0) 0.999 (1) 1.000 (2) 1.000 (3) 0.999 (4)
1.000 (5) 1.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 0.999 (11) 0.999 (12) 1.000 (13) 0.999 (14)
1.000 (15) 0.999 (16) 0.999 (17) 0.999 (18) 0.999 (19)
1.000 (20) 0.999 (21) 0.999 (22) 0.999 (23) 1.000 (24)
1.000 (25) 0.999 (26) 0.999 (27) 0.999 (28) 0.999 (29)
Average = 1.000 Range = 0.001

rule 184
0.600 (0) 0.600 (1) 0.600 (2) 0.600 (3) 0.600 (4)
0.600 (5) 0.600 (6) 0.600 (7) 0.600 (8) 0.600 (9)
0.600 (10) 0.600 (11) 0.600 (12) 0.600 (13) 0.600 (14)
0.600 (15) 0.600 (16) 0.600 (17) 0.600 (18) 0.600 (19)
0.600 (20) 0.600 (21) 0.600 (22) 0.600 (23) 0.600 (24)
0.600 (25) 0.600 (26) 0.600 (27) 0.600 (28) 0.600 (29)
Average = 0.600 Range = 0.001

rule 185
0.600 (0) 0.600 (1) 0.600 (2) 0.600 (3) 0.600 (4)
0.600 (5) 0.600 (6) 0.600 (7) 0.600 (8) 0.600 (9)
0.600 (10) 0.600 (11) 0.600 (12) 0.600 (13) 0.600 (14)
0.600 (15) 0.600 (16) 0.600 (17) 0.600 (18) 0.600 (19)
0.600 (20) 0.600 (21) 0.600 (22) 0.600 (23) 0.600 (24)
0.600 (25) 0.600 (26) 0.600 (27) 0.600 (28) 0.600 (29)
Average = 0.600 Range = 0.000

rule 186
0.633 (0) 0.633 (1) 0.633 (2) 0.633 (3) 0.633 (4)
0.633 (5) 0.633 (6) 0.633 (7) 0.633 (8) 0.633 (9)
0.633 (10) 0.633 (11) 0.633 (12) 0.633 (13) 0.633 (14)
0.633 (15) 0.633 (16) 0.633 (17) 0.633 (18) 0.633 (19)
0.633 (20) 0.633 (21) 0.633 (22) 0.633 (23) 0.633 (24)
0.633 (25) 0.633 (26) 0.633 (27) 0.633 (28) 0.633 (29)
Average = 0.633 Range = 0.000

rule 187
0.700 (0) 0.700 (1) 0.700 (2) 0.700 (3) 0.700 (4)
0.700 (5) 0.700 (6) 0.700 (7) 0.700 (8) 0.700 (9)
0.700 (10) 0.700 (11) 0.700 (12) 0.700 (13) 0.700 (14)
0.700 (15) 0.700 (16) 0.700 (17) 0.700 (18) 0.700 (19)
0.700 (20) 0.700 (21) 0.700 (22) 0.700 (23) 0.700 (24)
0.700 (25) 0.700 (26) 0.700 (27) 0.700 (28) 0.700 (29)
Average = 0.700 Range = 0.000

rule 188
0.800 (0) 0.800 (1) 0.800 (2) 0.800 (3) 0.800 (4)
0.800 (5) 0.800 (6) 0.800 (7) 0.800 (8) 0.800 (9)
0.800 (10) 0.800 (11) 0.800 (12) 0.800 (13) 0.800 (14)
0.800 (15) 0.800 (16) 0.800 (17) 0.800 (18) 0.800 (19)
0.800 (20) 0.800 (21) 0.800 (22) 0.800 (23) 0.800 (24)
0.800 (25) 0.800 (26) 0.800 (27) 0.800 (28) 0.800 (29)
Average = 0.800 Range = 0.000

rule 189
0.833 (0) 0.833 (1) 0.833 (2) 0.833 (3) 0.833 (4)
0.833 (5) 0.833 (6) 0.833 (7) 0.833 (8) 0.833 (9)
0.833 (10) 0.833 (11) 0.833 (12) 0.833 (13) 0.833 (14)
0.833 (15) 0.833 (16) 0.833 (17) 0.833 (18) 0.833 (19)
0.833 (20) 0.833 (21) 0.833 (22) 0.833 (23) 0.833 (24)
0.833 (25) 0.833 (26) 0.833 (27) 0.833 (28) 0.833 (29)
Average = 0.833 Range = 0.000

rule 190
0.800 (0) 0.800 (1) 0.800 (2) 0.800 (3) 0.800 (4)
0.800 (5) 0.800 (6) 0.800 (7) 0.800 (8) 0.800 (9)
0.800 (10) 0.800 (11) 0.800 (12) 0.800 (13) 0.800 (14)
0.800 (15) 0.800 (16) 0.800 (17) 0.800 (18) 0.800 (19)
0.800 (20) 0.800 (21) 0.800 (22) 0.800 (23) 0.800 (24)
0.800 (25) 0.800 (26) 0.800 (27) 0.800 (28) 0.800 (29)
Average = 0.800 Range = 0.000

rule 191
0.800 (0) 0.800 (1) 0.800 (2) 0.800 (3) 0.800 (4)
0.800 (5) 0.800 (6) 0.800 (7) 0.800 (8) 0.800 (9)
0.800 (10) 0.800 (11) 0.800 (12) 0.800 (13) 0.800 (14)
0.800 (15) 0.800 (16) 0.800 (17) 0.800 (18) 0.800 (19)
0.800 (20) 0.800 (21) 0.800 (22) 0.800 (23) 0.800 (24)
0.800 (25) 0.800 (26) 0.800 (27) 0.800 (28) 0.800 (29)
Average = 0.800 Range = 0.000

rule 192
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 0.000 (4)
0.000 (5) 0.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 0.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
0.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 0.000 (29)
Average = 0.000 Range = 0.000

rule 193
0.346 (0) 0.461 (1) 0.347 (2) 0.461 (3) 0.346 (4)
0.461 (5) 0.347 (6) 0.461 (7) 0.347 (8) 0.462 (9)
0.347 (10) 0.461 (11) 0.346 (12) 0.460 (13) 0.347 (14)
0.460 (15) 0.347 (16) 0.459 (17) 0.348 (18) 0.459 (19)
0.348 (20) 0.460 (21) 0.348 (22) 0.460 (23) 0.347 (24)
0.460 (25) 0.346 (26) 0.460 (27) 0.346 (28) 0.461 (29)
Average = 0.404 Range = 0.116

rule 194
0.167 (0) 0.167 (1) 0.167 (2) 0.167 (3) 0.167 (4)
0.167 (5) 0.167 (6) 0.167 (7) 0.167 (8) 0.167 (9)
0.167 (10) 0.167 (11) 0.167 (12) 0.167 (13) 0.167 (14)
0.167 (15) 0.167 (16) 0.167 (17) 0.167 (18) 0.167 (19)
0.167 (20) 0.167 (21) 0.167 (22) 0.167 (23) 0.167 (24)
0.167 (25) 0.167 (26) 0.167 (27) 0.167 (28) 0.167 (29)
Average = 0.167 Range = 0.000

rule 195
0.467 (0) 0.467 (1) 0.534 (2) 0.400 (3) 0.467 (4)
0.467 (5) 0.533 (6) 0.600 (7) 0.400 (8) 0.467 (9)
0.533 (10) 0.533 (11) 0.600 (12) 0.533 (13) 0.467 (14)
0.400 (15) 0.467 (16) 0.400 (17) 0.467 (18) 0.533 (19)
0.600 (20) 0.400 (21) 0.467 (22) 0.600 (23) 0.400 (24)
0.400 (25) 0.467 (26) 0.667 (27) 0.333 (28) 0.400 (29)
Average = 0.482 Range = 0.333

rule 196
0.000 (0) 1.000 (1) 0.000 (2) 1.000 (3) 0.000 (4)
0.000 (5) 1.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
1.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 1.000 (14)
0.000 (15) 0.000 (16) 1.000 (17) 0.000 (18) 0.000 (19)
0.000 (20) 0.000 (21) 1.000 (22) 0.000 (23) 0.000 (24)
0.000 (25) 1.000 (26) 0.000 (27) 1.000 (28) 0.000 (29)
Average = 0.300 Range = 1.000

rule 197
0.000 (0) 0.000 (1) 1.000 (2) 0.000 (3) 1.000 (4)
0.000 (5) 0.000 (6) 1.000 (7) 0.000 (8) 1.000 (9)
0.000 (10) 1.000 (11) 0.000 (12) 1.000 (13) 0.000 (14)
1.000 (15) 0.000 (16) 1.000 (17) 0.000 (18) 1.000 (19)
0.000 (20) 1.000 (21) 0.000 (22) 1.000 (23) 0.000 (24)
1.000 (25) 0.000 (26) 1.000 (27) 0.000 (28) 1.000 (29)
Average = 0.467 Range = 1.000

rule 198
1.000 (0) 0.000 (1) 0.500 (2) 1.000 (3) 0.000 (4)
0.500 (5) 1.000 (6) 0.000 (7) 0.500 (8) 1.000 (9)
0.000 (10) 1.000 (11) 0.000 (12) 1.000 (13) 0.000 (14)
1.000 (15) 0.000 (16) 1.000 (17) 0.000 (18) 1.000 (19)
0.000 (20) 0.500 (21) 1.000 (22) 0.000 (23) 0.500 (24)
1.000 (25) 0.000 (26) 0.500 (27) 1.000 (28) 0.000 (29)
Average = 0.500 Range = 1.000

rule 199
1.000 (0) 0.000 (1) 1.000 (2) 0.000 (3) 0.500 (4)
1.000 (5) 0.000 (6) 0.500 (7) 1.000 (8) 0.000 (9)
1.000 (10) 0.000 (11) 1.000 (12) 0.000 (13) 1.000 (14)
0.000 (15) 1.000 (16) 0.000 (17) 1.000 (18) 0.000 (19)
0.500 (20) 1.000 (21) 0.000 (22) 1.000 (23) 0.000 (24)
1.000 (25) 0.000 (26) 1.000 (27) 0.000 (28) 0.500 (29)
Average = 0.500 Range = 1.000

rule 200
1.000 (0) 1.000 (1) 0.000 (2) 0.000 (3) 0.000 (4)
1.000 (5) 1.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
0.000 (10) 0.000 (11) 1.000 (12) 1.000 (13) 1.000 (14)
0.000 (15) 1.000 (16) 1.000 (17) 0.000 (18) 0.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 0.000 (23) 0.000 (24)
1.000 (25) 1.000 (26) 0.000 (27) 0.000 (28) 0.000 (29)
Average = 0.467 Range = 1.000

rule 201
0.000 (0) 1.000 (1) 1.000 (2) 0.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
0.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 0.000 (14)
0.500 (15) 0.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 0.000 (22) 0.000 (23) 1.000 (24)
1.000 (25) 0.000 (26) 0.500 (27) 0.000 (28) 0.500 (29)
Average = 0.650 Range = 1.000

rule 202
1.000 (0) 0.000 (1) 1.000 (2) 1.000 (3) 1.000 (4)
1.000 (5) 0.000 (6) 1.000 (7) 1.000 (8) 0.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 0.000 (13) 1.000 (14)
1.000 (15) 1.000 (16) 0.000 (17) 0.999 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 0.000 (28) 1.000 (29)
Average = 0.800 Range = 1.000

rule 203
1.000 (0) 1.000 (1) 0.000 (2) 1.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 0.000 (7) 1.000 (8) 1.000 (9)
0.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 1.000 (14)
0.000 (15) 1.000 (16) 1.000 (17) 0.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 0.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 0.000 (27) 1.000 (28) 1.000 (29)
Average = 0.767 Range = 1.000

rule 204
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 1.000 (4)
0.000 (5) 0.000 (6) 1.000 (7) 0.000 (8) 1.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
1.000 (15) 0.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 0.000 (22) 0.000 (23) 1.000 (24)
1.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 1.000 (29)
Average = 0.400 Range = 1.000

rule 205
0.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 0.000 (4)
0.000 (5) 1.000 (6) 0.000 (7) 1.000 (8) 1.000 (9)
0.000 (10) 1.000 (11) 0.000 (12) 1.000 (13) 0.000 (14)
1.000 (15) 1.000 (16) 0.000 (17) 1.000 (18) 1.000 (19)
0.000 (20) 1.000 (21) 1.000 (22) 0.000 (23) 0.000 (24)
1.000 (25) 0.000 (26) 1.000 (27) 1.000 (28) 0.000 (29)
Average = 0.567 Range = 1.000

rule 206
1.000 (0) 0.000 (1) 1.000 (2) 0.000 (3) 1.000 (4)
1.000 (5) 0.000 (6) 1.000 (7) 1.000 (8) 0.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 0.000 (13) 1.000 (14)
1.000 (15) 1.000 (16) 0.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 0.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 0.000 (28) 1.000 (29)
Average = 0.733 Range = 1.000

rule 207
1.000 (0) 1.000 (1) 0.000 (2) 1.000 (3) 0.000 (4)
1.000 (5) 1.000 (6) 0.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 0.000 (11) 1.000 (12) 1.000 (13) 1.000 (14)
0.000 (15) 1.000 (16) 1.000 (17) 0.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 0.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 0.000 (27) 1.000 (28) 0.000 (29)
Average = 0.700 Range = 1.000

rule 208
0.333 (0) 0.333 (1) 0.333 (2) 0.333 (3) 0.333 (4)
0.333 (5) 0.333 (6) 0.333 (7) 0.333 (8) 0.333 (9)
0.333 (10) 0.333 (11) 0.333 (12) 0.333 (13) 0.333 (14)
0.333 (15) 0.333 (16) 0.333 (17) 0.333 (18) 0.333 (19)
0.333 (20) 0.333 (21) 0.333 (22) 0.333 (23) 0.334 (24)
0.334 (25) 0.334 (26) 0.334 (27) 0.333 (28) 0.333 (29)
Average = 0.333 Range = 0.001

rule 209
0.600 (0) 0.600 (1) 0.600 (2) 0.600 (3) 0.600 (4)
0.600 (5) 0.600 (6) 0.600 (7) 0.600 (8) 0.600 (9)
0.600 (10) 0.600 (11) 0.600 (12) 0.600 (13) 0.600 (14)
0.600 (15) 0.600 (16) 0.600 (17) 0.600 (18) 0.600 (19)
0.600 (20) 0.600 (21) 0.600 (22) 0.600 (23) 0.600 (24)
0.600 (25) 0.600 (26) 0.600 (27) 0.600 (28) 0.600 (29)
Average = 0.600 Range = 0.000

rule 210
0.567 (0) 0.534 (1) 0.567 (2) 0.533 (3) 0.567 (4)
0.533 (5) 0.567 (6) 0.533 (7) 0.567 (8) 0.533 (9)
0.567 (10) 0.533 (11) 0.567 (12) 0.533 (13) 0.567 (14)
0.533 (15) 0.567 (16) 0.533 (17) 0.567 (18) 0.533 (19)
0.567 (20) 0.533 (21) 0.567 (22) 0.533 (23) 0.567 (24)
0.533 (25) 0.567 (26) 0.534 (27) 0.567 (28) 0.533 (29)
Average = 0.550 Range = 0.034

rule 211
0.733 (0) 0.667 (1) 0.733 (2) 0.667 (3) 0.733 (4)
0.667 (5) 0.733 (6) 0.667 (7) 0.733 (8) 0.667 (9)
0.733 (10) 0.667 (11) 0.733 (12) 0.667 (13) 0.734 (14)
0.667 (15) 0.733 (16) 0.667 (17) 0.733 (18) 0.666 (19)
0.733 (20) 0.667 (21) 0.733 (22) 0.667 (23) 0.733 (24)
0.667 (25) 0.733 (26) 0.666 (27) 0.733 (28) 0.667 (29)
Average = 0.700 Range = 0.067

rule 212
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.500 (28) 0.500 (29)
Average = 0.500 Range = 0.000

rule 213
0.433 (0) 0.567 (1) 0.433 (2) 0.567 (3) 0.433 (4)
0.567 (5) 0.433 (6) 0.567 (7) 0.433 (8) 0.567 (9)
0.433 (10) 0.567 (11) 0.433 (12) 0.567 (13) 0.433 (14)
0.567 (15) 0.433 (16) 0.567 (17) 0.433 (18) 0.567 (19)
0.433 (20) 0.567 (21) 0.433 (22) 0.567 (23) 0.433 (24)
0.567 (25) 0.433 (26) 0.567 (27) 0.433 (28) 0.567 (29)
Average = 0.500 Range = 0.134

rule 214
0.767 (0) 0.733 (1) 0.767 (2) 0.733 (3) 0.767 (4)
0.733 (5) 0.767 (6) 0.733 (7) 0.767 (8) 0.733 (9)
0.767 (10) 0.733 (11) 0.767 (12) 0.733 (13) 0.767 (14)
0.733 (15) 0.767 (16) 0.733 (17) 0.766 (18) 0.733 (19)
0.766 (20) 0.733 (21) 0.766 (22) 0.733 (23) 0.766 (24)
0.733 (25) 0.766 (26) 0.733 (27) 0.766 (28) 0.733 (29)
Average = 0.750 Range = 0.034

rule 215
0.733 (0) 0.667 (1) 0.733 (2) 0.667 (3) 0.733 (4)
0.667 (5) 0.733 (6) 0.667 (7) 0.733 (8) 0.667 (9)
0.733 (10) 0.667 (11) 0.733 (12) 0.667 (13) 0.734 (14)
0.667 (15) 0.733 (16) 0.667 (17) 0.733 (18) 0.666 (19)
0.733 (20) 0.667 (21) 0.733 (22) 0.667 (23) 0.733 (24)
0.666 (25) 0.734 (26) 0.666 (27) 0.733 (28) 0.666 (29)
Average = 0.700 Range = 0.067

rule 216
1.000 (0) 0.999 (1) 0.999 (2) 0.999 (3) 0.999 (4)
0.999 (5) 0.999 (6) 0.999 (7) 0.999 (8) 0.999 (9)
0.999 (10) 0.999 (11) 0.999 (12) 0.998 (13) 0.998 (14)
0.998 (15) 0.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 0.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 1.000 (29)
Average = 0.933 Range = 1.000

rule 217
0.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 0.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 0.000 (13) 1.000 (14)
1.000 (15) 1.000 (16) 0.000 (17) 1.000 (18) 1.000 (19)
0.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 0.000 (26) 1.000 (27) 1.000 (28) 1.000 (29)
Average = 0.800 Range = 1.000

rule 218
1.000 (0) 1.000 (1) 1.000 (2) 0.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 1.000 (14)
0.000 (15) 1.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 0.000 (28) 1.000 (29)
Average = 0.900 Range = 1.000

rule 219
1.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 1.000 (14)
0.000 (15) 1.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 1.000 (29)
Average = 0.967 Range = 1.000

rule 220
1.000 (0) 1.000 (1) 1.000 (2) 0.000 (3) 1.000 (4)
1.000 (5) 0.000 (6) 1.000 (7) 0.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 0.000 (14)
1.000 (15) 0.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 0.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 0.000 (28) 1.000 (29)
Average = 0.767 Range = 1.000

rule 221
1.000 (0) 0.000 (1) 1.000 (2) 0.000 (3) 1.000 (4)
1.000 (5) 0.000 (6) 1.000 (7) 1.000 (8) 0.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 0.000 (13) 1.000 (14)
1.000 (15) 1.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
0.000 (20) 1.000 (21) 1.000 (22) 0.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 0.000 (28) 1.000 (29)
Average = 0.733 Range = 1.000

rule 222
1.000 (0) 0.000 (1) 1.000 (2) 0.000 (3) 1.000 (4)
1.000 (5) 0.000 (6) 1.000 (7) 1.000 (8) 0.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 0.000 (13) 1.000 (14)
1.000 (15) 1.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 0.000 (28) 1.000 (29)
Average = 0.800 Range = 1.000

rule 223
1.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 1.000 (7) 0.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 1.000 (14)
1.000 (15) 0.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 1.000 (29)
Average = 0.933 Range = 1.000

rule 224
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 0.000 (4)
0.000 (5) 0.000 (6) 0.000 (7) 0.000 (8) 0.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.000 (19)
0.001 (20) 0.001 (21) 0.000 (22) 0.000 (23) 0.000 (24)
0.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 0.000 (29)
Average = 0.000 Range = 0.001

rule 225
0.498 (0) 0.497 (1) 0.487 (2) 0.491 (3) 0.492 (4)
0.496 (5) 0.489 (6) 0.492 (7) 0.493 (8) 0.491 (9)
0.494 (10) 0.490 (11) 0.492 (12) 0.497 (13) 0.489 (14)
0.496 (15) 0.495 (16) 0.499 (17) 0.496 (18) 0.497 (19)
0.490 (20) 0.503 (21) 0.494 (22) 0.503 (23) 0.492 (24)
0.502 (25) 0.502 (26) 0.502 (27) 0.493 (28) 0.498 (29)
Average = 0.495 Range = 0.016

rule 226
0.600 (0) 0.600 (1) 0.600 (2) 0.600 (3) 0.600 (4)
0.600 (5) 0.600 (6) 0.600 (7) 0.600 (8) 0.600 (9)
0.600 (10) 0.600 (11) 0.600 (12) 0.600 (13) 0.600 (14)
0.600 (15) 0.600 (16) 0.600 (17) 0.600 (18) 0.600 (19)
0.600 (20) 0.600 (21) 0.600 (22) 0.600 (23) 0.600 (24)
0.600 (25) 0.600 (26) 0.600 (27) 0.600 (28) 0.600 (29)
Average = 0.600 Range = 0.000

rule 227
0.600 (0) 0.600 (1) 0.600 (2) 0.600 (3) 0.600 (4)
0.600 (5) 0.600 (6) 0.600 (7) 0.600 (8) 0.600 (9)
0.600 (10) 0.600 (11) 0.600 (12) 0.600 (13) 0.600 (14)
0.600 (15) 0.600 (16) 0.600 (17) 0.600 (18) 0.600 (19)
0.600 (20) 0.600 (21) 0.600 (22) 0.600 (23) 0.600 (24)
0.600 (25) 0.600 (26) 0.600 (27) 0.600 (28) 0.600 (29)
Average = 0.600 Range = 0.000

rule 228
0.000 (0) 0.000 (1) 0.000 (2) 0.000 (3) 1.000 (4)
0.000 (5) 0.000 (6) 0.000 (7) 0.000 (8) 1.000 (9)
0.000 (10) 0.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 0.000 (18) 0.001 (19)
0.001 (20) 1.000 (21) 0.000 (22) 0.000 (23) 0.000 (24)
1.000 (25) 0.000 (26) 0.000 (27) 0.000 (28) 1.000 (29)
Average = 0.167 Range = 1.000

rule 229
0.667 (0) 0.600 (1) 0.667 (2) 0.600 (3) 0.666 (4)
0.600 (5) 0.666 (6) 0.600 (7) 0.666 (8) 0.600 (9)
0.666 (10) 0.600 (11) 0.667 (12) 0.600 (13) 0.667 (14)
0.600 (15) 0.667 (16) 0.600 (17) 0.667 (18) 0.600 (19)
0.667 (20) 0.600 (21) 0.667 (22) 0.600 (23) 0.666 (24)
0.600 (25) 0.667 (26) 0.600 (27) 0.667 (28) 0.600 (29)
Average = 0.633 Range = 0.067

rule 230
0.833 (0) 0.833 (1) 0.833 (2) 0.833 (3) 0.833 (4)
0.833 (5) 0.833 (6) 0.833 (7) 0.833 (8) 0.833 (9)
0.833 (10) 0.833 (11) 0.833 (12) 0.833 (13) 0.833 (14)
0.833 (15) 0.833 (16) 0.833 (17) 0.833 (18) 0.833 (19)
0.833 (20) 0.833 (21) 0.833 (22) 0.833 (23) 0.833 (24)
0.833 (25) 0.833 (26) 0.833 (27) 0.833 (28) 0.833 (29)
Average = 0.833 Range = 0.000

rule 231
0.833 (0) 0.833 (1) 0.833 (2) 0.833 (3) 0.833 (4)
0.833 (5) 0.833 (6) 0.833 (7) 0.833 (8) 0.833 (9)
0.833 (10) 0.833 (11) 0.833 (12) 0.833 (13) 0.833 (14)
0.833 (15) 0.833 (16) 0.833 (17) 0.833 (18) 0.833 (19)
0.833 (20) 0.833 (21) 0.833 (22) 0.833 (23) 0.833 (24)
0.833 (25) 0.833 (26) 0.833 (27) 0.833 (28) 0.833 (29)
Average = 0.833 Range = 0.000

rule 232
0.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 0.000 (4)
0.000 (5) 0.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 0.000 (12) 0.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 0.000 (23) 0.000 (24)
0.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 0.000 (29)
Average = 0.533 Range = 1.000

rule 233
1.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 1.000 (14)
1.000 (15) 1.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 1.000 (29)
Average = 1.000 Range = 0.000

rule 234
1.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 1.000 (14)
1.000 (15) 1.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 1.000 (29)
Average = 1.000 Range = 0.000

rule 235
1.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 1.000 (14)
1.000 (15) 1.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 1.000 (29)
Average = 1.000 Range = 0.000

rule 236
0.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 0.000 (4)
0.000 (5) 1.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 0.000 (14)
0.000 (15) 0.000 (16) 0.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 0.000 (23) 0.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 0.000 (29)
Average = 0.667 Range = 1.000

rule 237
1.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 1.000 (14)
1.000 (15) 1.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 0.000 (22) 0.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 1.000 (29)
Average = 0.933 Range = 1.000

rule 238
1.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 1.000 (14)
1.000 (15) 1.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 1.000 (29)
Average = 1.000 Range = 0.000

rule 239
1.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 1.000 (14)
1.000 (15) 1.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 1.000 (29)
Average = 1.000 Range = 0.000

rule 240
0.500 (0) 0.500 (1) 0.500 (2) 0.500 (3) 0.500 (4)
0.500 (5) 0.500 (6) 0.500 (7) 0.500 (8) 0.500 (9)
0.500 (10) 0.500 (11) 0.500 (12) 0.500 (13) 0.500 (14)
0.500 (15) 0.500 (16) 0.500 (17) 0.500 (18) 0.500 (19)
0.500 (20) 0.500 (21) 0.500 (22) 0.500 (23) 0.500 (24)
0.500 (25) 0.500 (26) 0.500 (27) 0.500 (28) 0.500 (29)
Average = 0.500 Range = 0.000

rule 241
0.667 (0) 0.667 (1) 0.667 (2) 0.667 (3) 0.667 (4)
0.667 (5) 0.667 (6) 0.667 (7) 0.667 (8) 0.667 (9)
0.667 (10) 0.667 (11) 0.667 (12) 0.667 (13) 0.667 (14)
0.667 (15) 0.667 (16) 0.667 (17) 0.667 (18) 0.667 (19)
0.667 (20) 0.667 (21) 0.667 (22) 0.667 (23) 0.667 (24)
0.667 (25) 0.667 (26) 0.667 (27) 0.666 (28) 0.667 (29)
Average = 0.667 Range = 0.000

rule 242
0.667 (0) 0.667 (1) 0.667 (2) 0.667 (3) 0.667 (4)
0.667 (5) 0.667 (6) 0.667 (7) 0.666 (8) 0.667 (9)
0.667 (10) 0.666 (11) 0.667 (12) 0.667 (13) 0.667 (14)
0.667 (15) 0.667 (16) 0.667 (17) 0.667 (18) 0.667 (19)
0.667 (20) 0.667 (21) 0.667 (22) 0.667 (23) 0.667 (24)
0.667 (25) 0.667 (26) 0.667 (27) 0.667 (28) 0.667 (29)
Average = 0.667 Range = 0.000

rule 243
0.767 (0) 0.767 (1) 0.767 (2) 0.767 (3) 0.767 (4)
0.767 (5) 0.767 (6) 0.767 (7) 0.767 (8) 0.767 (9)
0.766 (10) 0.766 (11) 0.766 (12) 0.767 (13) 0.767 (14)
0.767 (15) 0.767 (16) 0.767 (17) 0.767 (18) 0.767 (19)
0.767 (20) 0.767 (21) 0.767 (22) 0.767 (23) 0.767 (24)
0.767 (25) 0.767 (26) 0.767 (27) 0.767 (28) 0.767 (29)
Average = 0.767 Range = 0.000

rule 244
0.633 (0) 0.633 (1) 0.633 (2) 0.633 (3) 0.633 (4)
0.633 (5) 0.633 (6) 0.633 (7) 0.633 (8) 0.633 (9)
0.633 (10) 0.633 (11) 0.633 (12) 0.633 (13) 0.633 (14)
0.633 (15) 0.633 (16) 0.633 (17) 0.633 (18) 0.633 (19)
0.633 (20) 0.633 (21) 0.633 (22) 0.633 (23) 0.633 (24)
0.633 (25) 0.633 (26) 0.633 (27) 0.633 (28) 0.633 (29)
Average = 0.633 Range = 0.000

rule 245
0.733 (0) 0.733 (1) 0.733 (2) 0.733 (3) 0.733 (4)
0.733 (5) 0.733 (6) 0.733 (7) 0.733 (8) 0.733 (9)
0.733 (10) 0.733 (11) 0.733 (12) 0.733 (13) 0.733 (14)
0.733 (15) 0.733 (16) 0.733 (17) 0.733 (18) 0.733 (19)
0.733 (20) 0.733 (21) 0.733 (22) 0.733 (23) 0.733 (24)
0.733 (25) 0.733 (26) 0.733 (27) 0.733 (28) 0.733 (29)
Average = 0.733 Range = 0.000

rule 246
0.767 (0) 0.767 (1) 0.767 (2) 0.767 (3) 0.767 (4)
0.767 (5) 0.767 (6) 0.767 (7) 0.766 (8) 0.767 (9)
0.767 (10) 0.767 (11) 0.767 (12) 0.767 (13) 0.767 (14)
0.767 (15) 0.767 (16) 0.767 (17) 0.767 (18) 0.767 (19)
0.767 (20) 0.767 (21) 0.767 (22) 0.767 (23) 0.767 (24)
0.767 (25) 0.767 (26) 0.767 (27) 0.767 (28) 0.767 (29)
Average = 0.767 Range = 0.000

rule 247
0.800 (0) 0.800 (1) 0.800 (2) 0.800 (3) 0.800 (4)
0.800 (5) 0.800 (6) 0.800 (7) 0.800 (8) 0.800 (9)
0.800 (10) 0.800 (11) 0.800 (12) 0.800 (13) 0.800 (14)
0.800 (15) 0.800 (16) 0.800 (17) 0.800 (18) 0.800 (19)
0.800 (20) 0.800 (21) 0.800 (22) 0.800 (23) 0.800 (24)
0.800 (25) 0.800 (26) 0.800 (27) 0.800 (28) 0.800 (29)
Average = 0.800 Range = 0.000

rule 248
1.000 (0) 0.999 (1) 0.999 (2) 0.999 (3) 0.999 (4)
0.999 (5) 0.999 (6) 0.999 (7) 0.999 (8) 0.999 (9)
0.999 (10) 0.999 (11) 0.999 (12) 0.999 (13) 0.999 (14)
0.999 (15) 1.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 1.000 (29)
Average = 1.000 Range = 0.001

rule 249
1.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 1.000 (14)
1.000 (15) 1.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 1.000 (29)
Average = 1.000 Range = 0.000

rule 250
1.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 1.000 (14)
1.000 (15) 1.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 1.000 (29)
Average = 1.000 Range = 0.000

rule 251
1.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 1.000 (14)
1.000 (15) 1.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 1.000 (29)
Average = 1.000 Range = 0.000

rule 252
1.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 0.999 (14)
1.000 (15) 1.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 1.000 (29)
Average = 1.000 Range = 0.001

rule 253
1.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 1.000 (14)
1.000 (15) 1.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 1.000 (29)
Average = 1.000 Range = 0.000

rule 254
1.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 1.000 (14)
1.000 (15) 1.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 1.000 (29)
Average = 1.000 Range = 0.000

rule 255
1.000 (0) 1.000 (1) 1.000 (2) 1.000 (3) 1.000 (4)
1.000 (5) 1.000 (6) 1.000 (7) 1.000 (8) 1.000 (9)
1.000 (10) 1.000 (11) 1.000 (12) 1.000 (13) 1.000 (14)
1.000 (15) 1.000 (16) 1.000 (17) 1.000 (18) 1.000 (19)
1.000 (20) 1.000 (21) 1.000 (22) 1.000 (23) 1.000 (24)
1.000 (25) 1.000 (26) 1.000 (27) 1.000 (28) 1.000 (29)
Average = 1.000 Range = 0.000