ARCHITECTURES FOR TWO-DIMENSIONAL LATTICE COMPUTATIONS WITH LINEAR SPEEDUP

Steven D. Kugelmass

(Thesis)

CS-TR-156-88

June 1988

Architectures for Two-Dimensional Lattice Computations with Linear Speedup

Steven D. Kugelmass

A Dissertation
Presented to the
Faculty of Princeton University
In Candidacy for the Degree
of Doctor of Philosophy

Recommended for Acceptance by the
Department of
Computer Science

June, 1988

Abstract

Many problems are characterized by the fact that they deal with data values distributed on a regular mesh, or *lattice*. They arise in a wide variety of applications such as image processing, computer vision, the solution of partial differential equations, and the simulation of cellular automata. This dissertation explores theoretical and practical questions in the design of massively parallel machines for lattice processing.

We analyze and compare two architectures that are efficient for lattice computations and are suitable for VLSI implementation: the linear array, and a block partitioned architecture proposed by Sternberg. These architectures have a property called *linear speedup*. That is, n processors of fixed size and cost provide n times the throughput of one processor on the same problem instance. We find that the linear pipelined array is the more attractive of the two architectures for a two-dimensional lattice machine, because of its great simplicity.

We next study the effect of clock skew as a possible limitation on the ultimate performance of large, globally synchronized multi-processing systems. We propose and analyze a probabilistic model for clock skew accumulation based on variations in buffer and wire delays. Our main result is that the expected skew grows as $O(\log N)$ in a system with N buffers, each of which contributes an independent, zero-mean, Gaussian skew. We also derive bounds on expected total clock skew when the skew in each stage depends on wire length, and the distribution system is embedded in the plane.

The remainder of this dissertation describes the design and construction of a prototype machine, called LGM-1 (for Lattice Gas Machine), for simulating the Frisch-Hasslacher-Pomeau (FHP) lattice-gas model for fluid flow. It consists of a one-dimensional pipeline of ten identical full-custom chips hosted by a Sun 3/160C workstation. The 64-pin DIP chips were fabricated by MOSIS in 3μ CMOS, and each contains more than 65,000 transistors. The chips themselves are capable of 14 million site-updates/sec/chip. The particular workstation host and interface limit the performance of LGM-1 to 7 million site-updates/sec, which is nevertheless about 60 times faster than a software simulation on the DEC VAX 8650.

for Ariela

Acknowledgements

I extend my deepest gratitude to my advisor, Professor Kenneth Steiglitz, who accepted me as his apprentice for the last three years. This work would not have possible without his support, and the quality would have suffered without his critical eye.

Professors Kai Li and Rafael Alonso are thanked for their critical and thorough reading of this dissertation. MOSIS provided the fabrication service for the VLSI chips for which I am also indebted.

One can never thank one's parents and family enough for the love, guidance, encouragement and support that they give. They have always encouraged me to achieve the most that I can and to never stop trying.

Ariela, my wife, has been my light, my partner and my friend. It is to her that this dissertation is lovingly dedicated.

Table of Contents

Abstract iii
Acknowledgements v
Table of Contents vi
List of Figures and Tables viii

Chapter 1 Introduction and Background **1**
1.1 History 1
1.2 The FHP Lattice Gas Model 2
1.3 Scattering Rules 4
1.4 Summary of Results 8
1.5 Previously Published Material 9

Chapter 2 Analysis of Lattice Architectures **10**
2.1 Introduction 10
2.2 A Paradigm For Lattice Computations 11
2.3 Serial Pipelined Architectures for Lattice Processing 12
2.4 Wide Serial Architecture 14
2.5 Sternberg Partitioned Architecture 15
2.6 Analysis and Comparison of WSA and SPA 17
2.7 Wide Serial Architecture 17
2.8 Sternberg Partitioned Architecture 20
2.9 Discussion 21
2.10 Summary 23

Chapter 3 A Probabilistic Model For Clock Skew **24**
3.1 Introduction 24
3.2 A General Model of Signal Distribution 25
3.3 Analysis and Upper Bounds 26
3.4 Examples 30
3.5 Metric-Free Tree 30
3.6 Metric Tree 33
3.7 Discussion 34
3.8 Conclusions 35
3.9 Acknowledgements 35

Chapter 4 Lattice Gas Cellular Automaton CMOS Chip **36**
4.1 Introduction 36
4.2 The Lattice and Data Encoding 37
4.3 Raster Scan 40
4.4 Architecture and Organization Overview 41
4.5 Hardware Implementation 43
4.6 Shift Registers 44
4.7 Neighborhood Generator 45
4.8 Update Processors 47
4.9 Testing Circuitry 47
4.10 Miscellany 48
4.11 Testing and Performance 48
4.12 Comparison to Other Machines 49

Chapter 5 LGM-1, A Prototype Lattice Gas Machine **50**
5.1 Overview 50
5.2 Hardware 51
5.3 Implementation Overview 52
5.4 Data Path 52
5.5 Control Path 52
5.6 Programming Model 53
5.7 Graphic Display 55
5.8 Performance Levels of Current System 55

Chapter 6 A Proposed Environment For Fast Prototyping of Linear Systolic Processors **67**
6.1 Overview 67
6.2 Hardware Specification 70
6.3 Software Specification 71
6.4 Physical Specification 71
6.5 Final Comments 72

Chapter 7 Conclusions and Future Work **74**
7.1 Summary and Conclusions 74
7.2 Next Steps 76

Bibliography **78**

Appendix 1 ..... **81**

List of Figures and Tables

Figure 1-1	Single site with links numbered	3
Figure 1-2	The hexagonal lattice	4
Figure 1-3	Particle collision rule classes	5
Figure 1-4	Torroidal Lattice	7
Figure 2-1	One-dimensional pipeline	12
Figure 2-2	Hexagonal neighborhood	13
Figure 2-3	Wide serial architecture	15
Figure 2-4	Sternberg partitioned architecture	16
Figure 3-1	Clock distribution tree	30
Figure 3-2	Range/Skew of Random Variables (Metric Free)	32
Figure 3-3	Skew in tree with uniform and gaussian delay	32
Figure 3-4	Range/Skew of Random Variables (Metric)	34
Figure 4-1	Space/Time Diagram of Pipelined Computation	37
Figure 4-2	Theoretical and Implemented Lattices	38
Figure 4-3	Site state encoding	39
Figure 4-4	Lattice Site Numbering	40
Figure 4-5	Raster Scan Pattern	41
Figure 4-6	Basic Architectural Plan	42
Figure 4-7	Chip Floor Plan	44
Figure 4-8	Shift Register Data Flow	45
Figure 4-9	Neighborhood Generator Data Dependencies	46
Figure 4-10	Chip Timing Diagram	48
Figure 5-1	Basic Prototype Architectural Plan	51
Table 5-1	Reported Actual FHP Performance	65
Table 6-1	Software Entry Points	71

Chapter 1
Introduction and Background

1.1. History

Many problems are characterized by the fact that they deal with data values distributed on a regular mesh or *lattice*. They arise in a wide variety of applications such as image processing, computer vision, physical modeling, and the solution of partial differential equations. These problems often require tremendous computational resources and they are often solved on large supercomputers, capable of high-speed floating-point performance. However, because of their distributed nature, special purpose SIMD and MIMD processor systems have been built and applied to these problems [1-4].

Another, much simpler type of lattice computation received study in the early days of computer science: cellular automata (CA) [5]. They are thought to be able to mimic the way that computations are performed in nature because they model the simultaneous interaction of simple things distributed over a large area or volume [6].

A cellular automaton is built from the following components:

1) Discrete state space Σ over $0 \cdots k-1$.

2) Set of sites called domain Δ. This is a discrete regular lattice in n dimensions. Each site in the domain has a value from Σ and this is referred to as the state of the site. The state of all the sites taken collectively is called the state of the CA.

3) An update rule Φ gives the state at time $t+1$ from the state at time t using the state of sites from a fixed neighborhood over the n-dimensional lattice. Φ can therefore be thought of as a local re-writing rule.

The cellular automaton originally served as a model to study self-replication in systems, and efforts were directed toward understanding the structure of the computations it supported. After a while, the interest within the computer science community died down, partly because the cellular automaton came to be viewed in the less serious light of a mathematical game and a lay scientific recreation. Stephen Wolfram, in 1984 [7], reopened this field of research with a series of papers in which he undertook a study of some properties of a class of one-dimensional cellular automata. Vastly

increased computational power made this possible. The scientific community responded with a flurry of papers on the applications of CA to real problems and the beginning of a mathematical treatment of their important properties. Today, there are conferences dedicated to the subject of cellular automata and their applications [8-15].

Within two years following Wolfram's 1984 papers, an idea from the mid 1960s [16] was revived by a group of researchers (Frisch, Hasslacher, and Pomeau) at Los Alamos National Laboratories [17]. In their paper they showed that in the limit of large lattice size, their automaton converges to the two-dimensional Navier-Stokes equations. They simulated their discrete particle model of a fluid by computer program and the results of the simulation bore a striking similarity to photographs of actual flow visualization studies [18]. This was perhaps the earliest concrete example of where a cellular automaton might offer a competitive alternative method of solving a hard problem in high demand: fluid dynamics. This problem had traditionally been attacked by various grid and finite element techniques in the solution of the Navier-Stokes partial differential equations [19-21].

Frisch, Hasslacher and Pomeau's demonstration sparked our interest in lattice processing architectures because it was generally held at the time that only supercomputers would offer enough computational power in the simulations of CAs to allow anything but trivial examples. We sought, as a continuation of an informal discussion group on cellular automata and an ongoing VLSI project, an architecture for high performance lattice computations that exploited custom VLSI.

This dissertation addresses the general problem of designing a high-performance VLSI-based architecture for lattice computations. We use the Frisch, Hasslacher and Pomeau (FHP) lattice gas cellular automaton, briefly described in the next section, and originally presented in [17] as the touchstone for our work because it comprises many of the properties that are characteristic of lattice computations: small storage per site, large number of sites, small neighborhood per site, and complete uniformity of the algorithm. However, unlike other special purpose architectures dedicated to problems of this genre [22-25], our architecture can, in principle, be indefinitely pipelined and extended to provide arbitrary performance on a given problem: it has the property we call *linear speedup*.

1.2. The FHP Lattice Gas Model

The FHP lattice gas model is a discrete particle model for fluid dynamics [17]. It is based on a regular hexagonal lattice, giving each site six nearest neighbors. Particles move around the lattice, colliding only at lattice sites, and possibly scattering as a result. The links along which they move are bidirectional, and particles can interact only at lattice sites. Two particles traveling in opposite directions on the same link are not considered to be colliding.

Each site has six neighbors, and outgoing particles may be present on any of these links. See Figure 1-1. Therefore each site in the lattice has 64 states. However, some models must also provide for the possibility that there is a particle at rest at the site (a "center") and, additionally, that the site

may not be part of free space, but rather may be solid. The "solid" sites are used to compose boundaries and obstacles for the simulation. This raises the minimum number of bits required to describe the state of a site from six to eight.

The links of a site are numbered:

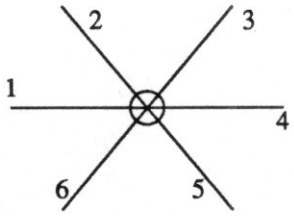


Figure 1-1: A single site with the links numbered.

The "B" bit denotes that the site is solid and the "C" bit denotes that the site contains a particle at rest, a "center."

The sites in the lattice are arranged in a hexagonal lattice, as illustrated in the following diagram.

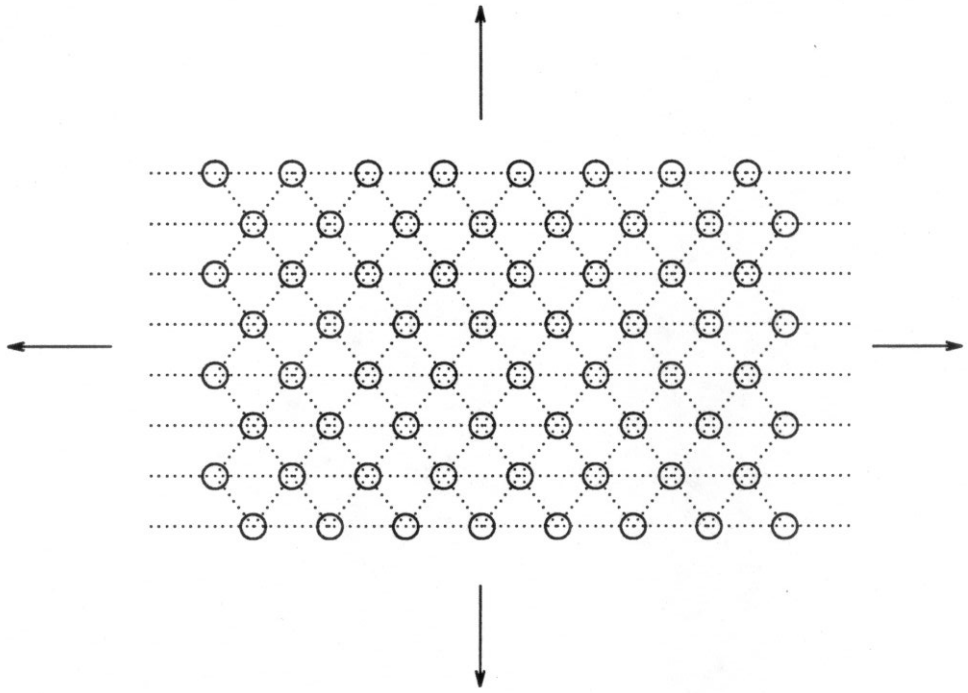


Figure 1-2: The hexagonal lattice.

1.3. Scattering Rules

The particles moving in the lattice scatter at the sites according to rules that are designed to make the simulation converge to solutions of the Navier-Stokes equations. The rules have two fundamental properties: conservation of mass (particle number), and conservation of momentum. Energy need not be conserved, and in fact, this extra conservation law is not desirable because it would inhibit the damping of oscillations and instabilities by the viscosity of the fluid.

Further, the collision rules are designed to minimize a particle's mean-free-path: the number of lattice edges it traverses without colliding and scattering with another particle. The smaller the mean-free-path, the lower the viscosity of the lattice gas, and the larger the Reynolds number of the simulation. This fact lends impetus to having as many particle interactions (scattering rules) as possible. In the event that a particle cannot scatter with another when they reach a lattice site, it continues its trajectory with its previous velocity undisturbed.

The following diagram (Figure 1-3) illustrates the particle collision rules. The configuration before the collision is shown on the left, with the configuration after the collision on the right. In the case where symmetry implies the existence of more than one member of a rule family, only one

member is shown. The complete set of rules can be derived by rotation of the before and after configurations by any multiple of 60 degrees. The small circle at the site means that a particle is present there at rest.

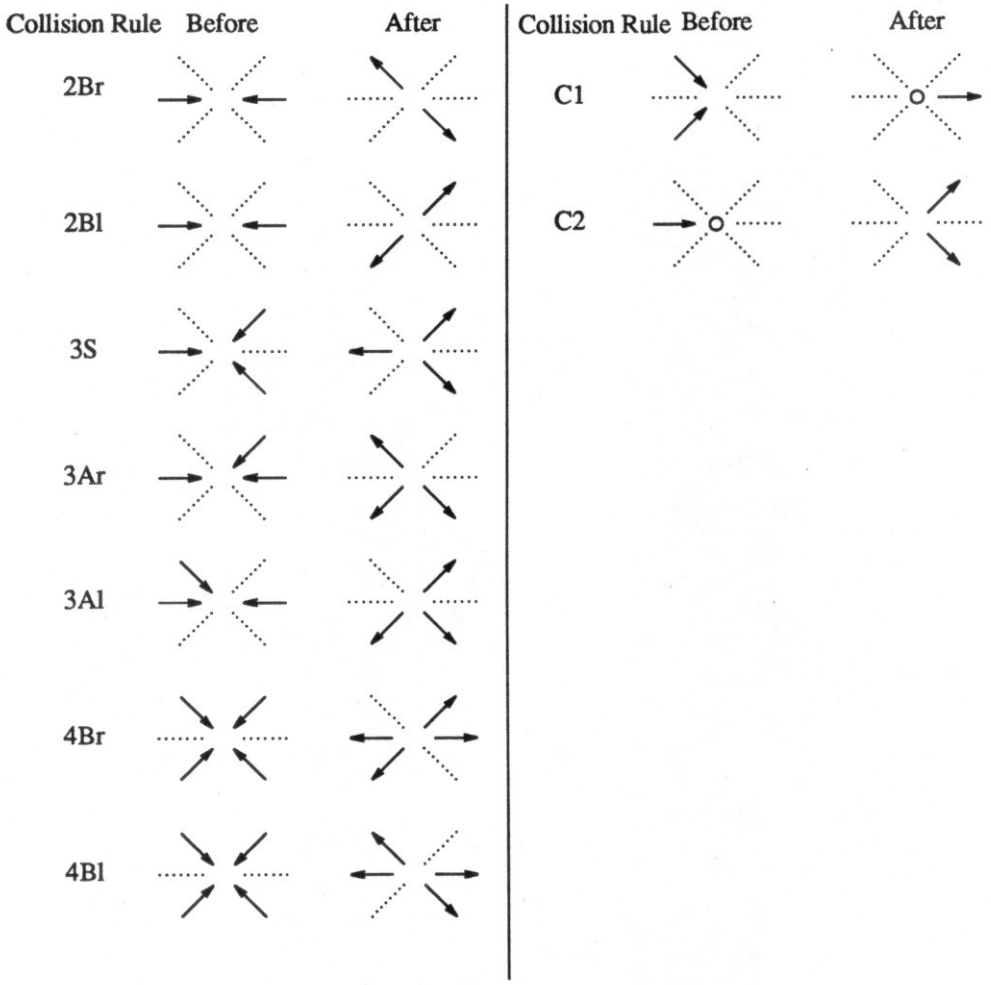


Figure 1-3: Particle collision rule classes

Obstacles, such as cylinders, plates, or wings, are placed in the lattice by changing a site from "free-space" to "solid." While a "free-space" site scatters particles according to the rules outlined above, a "solid" site behaves differently and there are several permissible choices. The simplest possibility is to have each particle reflect back along its incoming path. Another possibility is to have each particle reflect in such a way that its incident angle equals its reflected angle. Unfortunately, the latter entails having different kinds of solid sites, depending on position in an object.

These two choices also correspond to different kinds of boundary layer effects [17]. In the first case, the rule leads to a "no-slip" boundary, where the velocity of the fluid near a surface is zero. The

second case corresponds to a ''slip'' condition; the fluid next to a boundary can move. We adopt the ''no-slip'' boundary because it is simpler, and more realistic for the kinds of fluid dynamic phenomena that we expect to simulate.

Three of the rule classes, 2B, 3A, and 4B have L and R variants, so called because they scatter left-ward and right-ward respectively. It is critical that both of these scatterings take place with equal probability or a systematic bias will be introduced that will destroy the convergence properties of the simulation. When the simulation is performed in software, it is a simple matter to ''flip a coin'' and choose either an L or an R rule when appropriate. Alternatively, the software simulation can use the R rule during even-numbered generations and the L rule during the odd-numbered ones (or vice-versa). In hardware, however, it is undesirable to ''flip a coin'' or to change rules on every generation. Instead, one of the chip's two processors performs the R rules all the time, and the other performs the L rules. This has the effect of placing the L and R rules throughout the lattice in a manner similar to the squares of a checkerboard. The unbiased spatial distribution is equivalent (informally) to an unbiased temporal distribution (coin flipping) because the events of interest occur randomly throughout the lattice and the simulation.

The sequence in which particle collisions are computed at lattice sites is also critical to the success of the simulation. In concept, the entire lattice is updated simultaneously, with each update called a *generation*. In practice, the new site values can be computed in any sequence provided that particles from generation i are the only ones used to compute particle velocities for generation i+1. If this rule is violated, it is easy to give a configuration and a site update sequence that violates the mass (and therefore the momentum) conservation law [17].

The behavior of a single particle in the lattice is effectively random, but it is correlated with the behavior of other nearby particles. The speed of any particle in the lattice is unity, but the simulation is intended to model a continuous process. The net velocity of an ensemble of particles over a small region of the lattice (spatial averages) yields the hydrodynamic behavior that is sought. This function is not computed by the Lattice Gas Cellular Automaton (LGCA) chip; it is done as part of a post-processing phase after computation has taken place.

One question that remains before processing can be fully understood is: What is done with the edges of the lattice? Again, there are several possibilities. The edges can be pair-wise attached so that the lattice has the structure of a torus (Figure 1-4). This has the effect of replicating the simulation an infinite number of times in all directions.

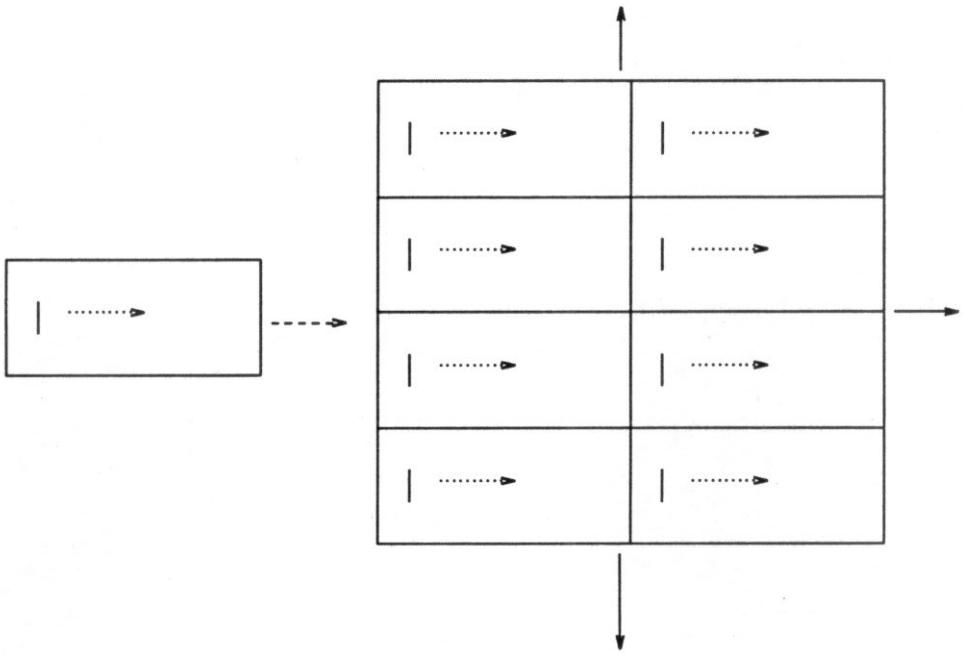


Figure 1-4: Toroidal Lattice

In some cases this mirroring could be considered undesirable. It might be preferable to make the finite lattice appear as if it were part of an infinite plane of fluid. Under certain conditions, this is possible. The key idea is that the links on the edges of the lattice should "see" incoming particles distributed as though the fluid extended infinitely in all directions, with a uniform velocity field. Let the average density of particles in the lattice be d. Put another way, d is the probability that a particle is found on a given link at any site in the lattice. Assume that there is a uniform velocity field $\vec{v}$, and that the links point in directions $\vec{a}_1, \ldots, \vec{a}_6$. It can be shown that the probability of finding a particle on link i is approximately [26]

$$P_i = d\left[1 + \frac{7}{3}\vec{v}\cdot\vec{a}_i - \left[\frac{7}{6}\frac{1-2d}{1-d}\left[\vec{v}^2 - \frac{7}{3}(\vec{a}_i\cdot\vec{v})^2\right]\right]\right] \tag{1-1}$$

The probability of finding a particle at rest at a site is given by Equation (1-1) with $\vec{a}_i\cdot\vec{v} = 0$.

If particles are introduced at the edges of the lattice with probabilities given by Equation (1-1), then the lattice appears to be embedded in an infinite plane of fluid which is moving at velocity $\vec{v}$.

1.4. Summary of Results

This dissertation is divided into two parts. Part I (Chapters 2 and 3) discusses the theoretical and analytical underpinnings of this work. Chapter 2 presents an analysis technique for determining the design space of architectures geared for lattice computations and based in VLSI. We derive the allowable design space based on the input/output limitation of the VLSI technology.

The architecture that we implemented can theoretically provide an arbitrarily high site update rate by concatenating any number of chips into a single pipeline. However, there is the question of whether or not this can actually be done in practice. In the course of developing the custom chips and the rest of the lattice gas system, we posed the following question: Can a linear pipeline be made arbitrarily deep or is there some point at which linear speedup breaks down and becomes sub-linear? Two factors that seemed reasonable candidates for causes of failure are timing and reliability.

The first, timing, has been studied extensively. Numerous clocking schemes are known and questions of synchronization and synchronization failure have been examined. However, there was little information available on a model that describes how problems might accumulate as systems increase in size. In fact, the only paper which we found that began to address such questions was [27].

Chapter 3 proposes and analyzes a probabilistic model for the accumulation of clock skew in large, synchronous, multiple processing element systems. Using this model, we derive upper bounds for expected skew, and its variance, in tree distribution systems with N synchronously clocked processing elements.

We apply these results to two specific models for clock distribution. In the first, which we call *metric-free*, the skew in a buffer stage is Gaussian with a variance independent of wire length. In this case the upper bound on skew grows as $\Theta(\log N)$ for a system with N processing elements.

The second, *metric*, model, is intended to reflect VLSI constraints: the clock skew in a stage is Gaussian with a variance proportional to wire length, and the distribution tree is an H-tree embedded in the plane. In this case the upper bound on expected skew is $\Theta(\sqrt{N \log N})$ for a system with N processors. Thus the probabilistic model is more optimistic than the deterministic summation model of Fisher and Kung [27], which predicts a clock skew $\Theta(N)$ in this case, and is also consistent with their lower bound of $\Omega(\sqrt{N})$ for planar embeddings.

We have estimates of the constants of proportionality, as well as the asymptotic behavior, and we have verified the accuracy of our estimates by simulation. This shows that clock skew grows as a function of system size, and system timing can limit the ultimate size of synchronous, two dimensionally organized processor systems.

Part II (Chapters 4 through 6) discusses the practical experience of implementing a system based on a custom CMOS chip that simulates the FHP lattice gas. Chapter 4 describes the architecture and implementation of the Lattice Gas Cellular Automaton (LGCA) chip which is a CMOS VLSI

processor that provides 14 million site updates per second. The chips can be pipelined for even higher performance, and it is theoretically possible to build a special purpose device that can provide the computational equivalent of a CRAY XMP-48 supercomputer or a Thinking Machines, Inc. Connection Machine for this problem.

To demonstrate the feasibility of such an undertaking, we have constructed a prototype, called LGM-1, which uses ten of these chips arranged in a pipeline. The LGM-1 computes approximately 7 million site updates per second, which is only 5% of the maximum throughput of the 10 VLSI chips. The discrepancy in performance levels is caused by the simple, low-bandwidth system interface and the relatively small power of the Sun 3 workstation host.

Chapter 5 describes the hardware that we built to allow a Sun 3 workstation to act as a host for our special purpose processor pipeline. Having had this experience, we gave some thought to the prototyping process as a whole. This leads us to Chapter 6, which describes a proposed prototyping environment which is sufficiently general to support chips fabricated by other designers. Its basic philosophy is that it is more profitable to be able to add hardware easily than it is to make the underlying system as fast as possible.

We conclude with Chapter 7, which summarizes and reviews the important results of this work. It also outlines future tasks for the continuation of the work begun with this thesis.

1.5. Previously Published Material

Chapter 2 is taken from the first half of a paper that was co-authored with Richard Squier and Kenneth Steiglitz and appears in its entirety in [28]. This work was supported in part by NSF Grant ECS-8414674, U. S. Army Research Office-Durham Contract DAAG29-85-K-0191, and DARPA Contract N00014-82-K-0549.

The results in Chapter 3 have been accepted for presentation at the International Conference on Systolic Arrays to be held in San Diego, California on May 25-27, 1988 and have been submitted for publication in the *IEEE Transactions on Computers*. This work was supported by NSF Grant MIP-8705454, U. S. Army Research Office-Durham Contract DAAG29-85-K-0191.

The material in Chapters 4 and 5 has been presented at the 22nd Conference on Information Sciences and Systems held in Princeton, NJ on March 16-18, 1988.

Chapter 2
Analysis of Lattice Architectures

2.1. Introduction

This chapter deals with the problems of designing and building practical, custom VLSI-based computers for lattice calculations. These computational problems are characterized by being iterative, defined on a regular lattice of points, uniform in space and time, local, and relatively simple at each lattice point. Examples include numerical solution of differential equations, iterative image processing, and cellular automata. The recently studied lattice gas automata, which are microscopic models for fluid dynamics, are proposed as a testbed for the work.

The machines envisaged — lattice engines — would typically consist of many instances of a custom chip and a general-purpose host machine for support. In many practical situations, the performance of such machines is limited, not by the speed and size of the actual processing elements, but by the communication bandwidth on- and off-chip, and by the memory capacity of the chip.

A familiar example of lattice-based computational tasks is two-dimensional image processing. Many useful algorithms, such as linear filtering and median filtering, recompute values the same way everywhere on the image, and so are perfectly uniform; they are local in that the computation at a given point depends only on the immediate neighbors of the point in the two-dimensional image.

Another class of calculations, besides being uniform and local, have the additional important characteristic of using only a few bits to store the values at lattice points, and so are extremely simple. Further, they operate on local data iteratively, which means that they are not as demanding of external data as many signal processing problems. These computational models — uniform, local, simple, and iterative — are called *cellular automata*. We will next describe a particular class of cellular automata, one that provides a good testbed for the general problems arising in the design of dedicated hardware for lattice-based computations.

2.2. A paradigm for lattice computations: the lattice gas model

Quite recently, there has been much attention given to a particularly promising kind of cellular automaton, the so-called *lattice gases*, because they can model fluid dynamics [29]. These are lattices governed by the following rules:

- At each lattice site, each edge of the lattice incident to that site may have exactly zero or one particle traveling at unit speed away from that site, and, in some models, possibly a particle at rest at the lattice site.
- There is a set of collision rules which determines, at each lattice site and at each time step, what the next particle configuration will be on its incident edges.
- The collision rules satisfy certain physically plausible laws, especially particle-number (mass) conservation, and momentum conservation.

These lattice gas models have an intrinsic exclusion principle, because no more than one particle can occupy a given directed lattice edge at any given time. It is therefore surprising that they can model fluid mechanics. In fact, in a two-dimensional hexagonally connected lattice, it has been shown that the Navier-Stokes equation is satisfied in the limit of large lattice size. This model is called the FHP model, after Frisch, Hasslacher, and Pomeau [17]. The older HPP [30] model, which uses an orthogonal lattice, does not lead to isotropic solutions [17].

The idea of using hexagonal lattice gas models to predict features of fluid flow seems to be about two years old, and whether the general approach of simulating a lattice gas can ever be competitive with more familiar numerical solution of the Navier-Stokes equation is certainly a premature question. Extensions to three dimensional gases are just now being formulated [31], and quantitative experimental verification of the two-dimensional results is fragmentary. The Reynolds numbers achievable depend on the size of the lattices used, and very large Reynolds numbers will require huge lattices, and correspondingly huge computation rates. For a discussion of the scaling of the lattice computations with Reynolds number, see [32].

What is clear is that the ultimate practicality of the approach will depend on the technology of special-purpose hardware implementations for the models involved. Furthermore, the *uniformity*, *locality* and *simplicity* of the model mean that this is an ideal testbed for dedicated hardware that is based on custom chips. We will therefore use the lattice gas problem as a running example in what follows. We especially want to study the interaction between the design of custom VLSI chips and the design of the overall system architecture for this class of problems.

We will present and compare two competing architectures for lattice gas cellular automata (LGCA) computations that are each based on VLSI custom processors. The analysis will focus on the permissible design space given the usual chip constraints of area and pin-out and on the achievable performance within the design space.

2.3. Serial Pipelined Architectures for Lattice Processing

We are primarily interested in special-purpose, VLSI-based processor architectures that have more than one PE (*processing element*) per custom chip. It is important to recognize that if the PE's are not kept busy, then it might be more effective (in terms of overall throughput) to have fewer PE's per chip but to use them more efficiently. Although there are many architectures that have the property of using PE's efficiently, we will describe only two, both based on the idea of serial pipelining (see Figure 2-1). The boxes containing the letter f are processing elements that compute the function f. The other boxes represent memory that is local to the processor.

This approach has the benefit that the bandwidth to the processor system is small even though the number of active PE's is large. The serial technique has been used for image processing where the size of the two-dimensional grid is small and fixed [11, 33, 34], and has also been used to design a high performance custom processor for a one-dimensional cellular automaton [35].

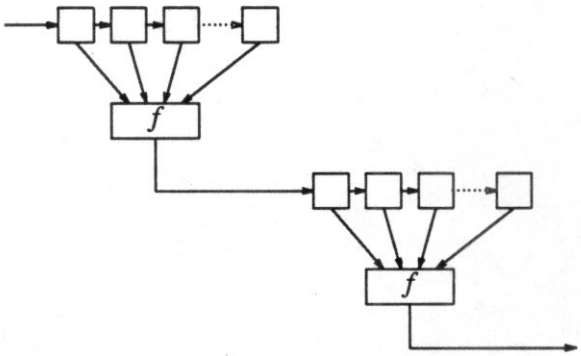


Figure 2-1: One-Dimensional Pipeline

Consider what is required to pipeline a computation. We must guarantee that the appropriate site values of the correct ages are presented to the computing elements. In the case of the LGCA, we can express this data dependency as:

$$a_i^{t+1} = f(N(a_i^t))$$

where a_i^t is the value at lattice site a_i at time t, $N(a_i^t)$ is the *neighborhood* of the lattice site a_i at time t, and f is the function that determines the new value of a_i based on its neighborhood. The LGCA requires all the points in the neighborhood of a to be the same age in order to compute the new value, a_i^{t+1}. The LGCA has a neighborhood that looks like:

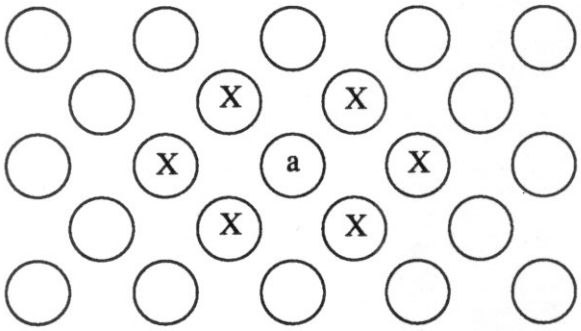


Figure 2-2: Hexagonal Neighborhood

The sites marked with X constitute the neighborhood of site *a*.

One-dimensional pipelining also requires a linear ordering of the sites in the array. That is, we wish to send the values associated with the sites one at a time into the one dimensional pipeline and receive the sequence of sites in the same order possibly some generations later. Therefore, we would like sites that are close together in the lattice to be close together in the stream. In this way, the serial PE requires a small local memory because neighborhoods (sites that are close together in the array) will also be close together in the stream. Unfortunately, the Lattice Gas Automaton can require a large amount of local memory per PE because there is no sub-linear embedding of an array into a list [36].

The natural row-major (raster scan) embedding of the array into a list preserves 2-neighborhoods* with diameter $2n-2$. This means that a full neighborhood of a site from an $n \times n$ lattice is distributed in the list so that some elements of the neighborhood are at least $2n-2$ positions apart. This embedding is undesirable for two reasons. The amount of local memory required by a PE is a function of the problem instance, forcing us to decide in advance the size of one dimension of the lattice (one can actually process a prism array, finite in all but one dimension) because the chip will only work for a single problem size due to its fixed span. The second deficiency is due to the size of the span. If n=1000, then each PE would require about 2000 sites worth of memory. This puts a severe restriction on the number of PE's that can be placed on a chip.

Unfortunately, the $2n-2$ embedding is optimal. Rosenberg showed this bound holds for prism array realizations but it has been unknown whether it is possible do better for finite array realizations. Rosenberg's best lower bound for the finite array case has never been achieved and he suspected that the row-major scheme was optimal. Sternberg [37] also questioned whether or not the storage requirement for a serial pipelined machine could be reduced. Supowit and Young [38] showed that the row-major embedding is optimal and therefore a serial pipeline must use at least $2n-2$ storage.

*Sites that are two edge traversals apart in the lattice

Theorem 2-1: Place the numbers $1, \ldots, n^2$ in a square array $a(i,j)$, and define the *span* of the array to be

$$\max\left\{ \; |a(i+1,j) - a(i,j)| \; , \; |a(i,j+1) - a(i,j)| \; \right\}$$

Then span $\geq n$.

Proof: Put the numbers in the array one at a time in order, starting with 1. When, for the first time, there is either a number in every row or a number in every column, stop. Without loss of generality, assume this happens with a number in every row.

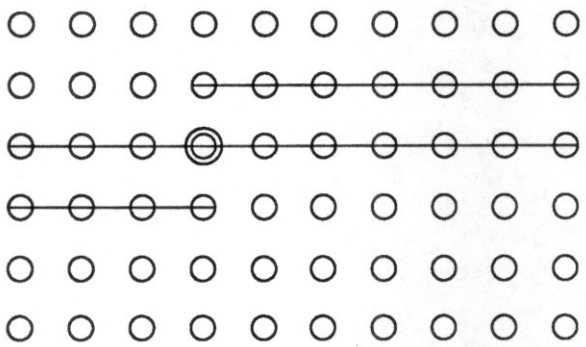
We claim that there cannot be a full row. Suppose the contrary. The last number entered was placed in an empty row, so there must have been a full row before we stopped. This would mean there was a number in every column before there was a number in every row.

Since there is no full row, but a number in every row, there is at least one vacant place in every row that is adjacent to an occupied spot. Choose one such vacant place in each row, and call them set F (with $|F| = n$). Now if we stopped after placing the number t, the places in F will get filled with numbers greater than t. The largest number that will be put in a location in F is $\geq t+n$, and will be adjacent to a number $\leq t$. □

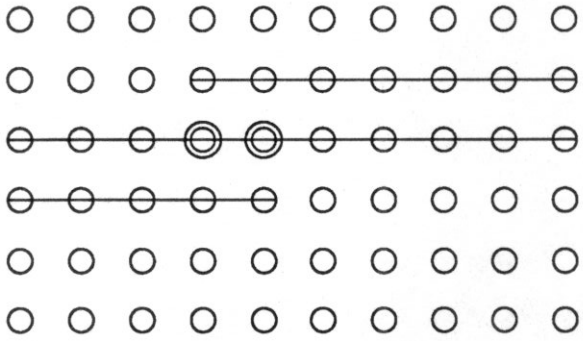
The appealing aspects of the serial architecture are the simplicity of its design, its small area in comparison to other architectures, and the small input/output bandwidth requirement. The computation proceeds on a wavefront [39] through time and space, each succeeding PE using the data from the previous PE without the need for further external data.

2.4. Wide Serial Architecture (WSA)

Assume that data from the two dimensional array is serialized by a row major raster scan. Throughput in a serial architecture can be improved by adding concurrency at each level of the pipeline. One way to accomplish this is to have each pipeline stage compute the new value of more than one site each clock period. For example, if the computation at PE j is at the point where site a, circled, is to be updated, then PE j contains the data indicated by strike-out in the following:



We could allow a second PE j' to compute site $a+1$ at the same time if we store just one more data point.



The most attractive feature of this scheme is that performance is increased, but at a cost of only the incremental amount of memory needed to store the extra sites. The on-chip memory per PE is also reduced dramatically, decreasing linearly with the number of PE's per chip. However, there is a price to pay: two new site values are required every clock period so that two site updates can be performed. The extra PE's require added bandwidth to and from the chip and this implies that the main memory system must provide that bandwidth as pins or wires.

As an example, the following figure shows how two PE's on the same chip can cooperate on a computation. Each square of the shift register memory holds the value of one site in the lattice. Every clock period, two new site values are input to the chip, two sites updated and their values output to the next chip in the pipeline.

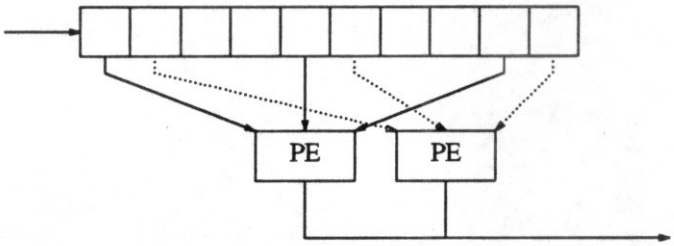


Figure 2-3: Wide Serial Architecture

2.5. Sternberg Partitioned Architecture (SPA)

In [37] Sternberg proposes that a large array computation can be divided among several serial processors, each of which operates as described earlier. The array is divided into adjacent, non-overlapping columnar slices and a fully serial processor is assigned to each slice (see Figure 2-4).

The processors are not exactly the same as those described above; they are augmented to provide a bidirectional synchronous communication channel between adjacent partitions so that sites

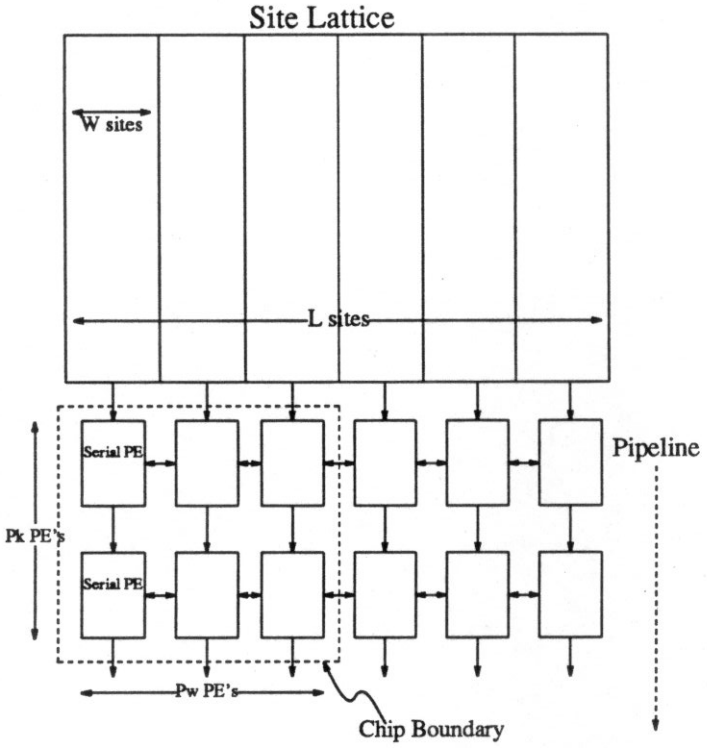


Figure 2-4: Sternberg Partitioned Architecture

whose neighborhoods do not lie entirely in the storage of a single PE can be computed correctly and in step with other site updates. See [37] for details.

Dividing the work in this way accomplishes three things. First, it decreases the amount of storage that each PE needs in order to delay site values for correct operation of the pipeline. This comes about because each PE needs to delay only two lines of its slice, not of the whole line width. Second, it increases the ratio of processing elements to the total number of sites, permitting an increase in the maximum throughput by a multiplicative constant equal to the number of slices. Third, it provides a degree of modularity and extensibility. It is possible to join two smaller machines along an edge to form a machine that handles a larger problem.

In the case of a VLSI implementation, decreasing the size of the local storage is extremely important because most of the silicon area in the implementation of a serial processor is shift register. Since each PE in the **SPA** architecture requires fewer shift register storage cells, it is possible to place several PE's on a chip, whereas if each serial PE were required to store two lines of the whole lattice, then only one or two PE's could be placed on a VLSI chip with current technology. The only way around this limitation is to use another technology to implement the required storage, such as off-chip commercial memories, in which case we quickly encounter pin limitations.

It is important to recognize that the *total* amount of storage required under this organization is two lines of the whole lattice per pipeline stage. Thus the total storage requirement under this implementation is not reduced below that of the fully serial approach presented earlier. In addition, each column of serial processors requires its own data path to and from main memory, and this data path is a relatively expensive commodity.

2.6. Analysis and Comparison of WSA and SPA

In this section we analyze and compare the Sternberg partitioned architecture (**SPA**) with the wide-serial architecture (**WSA**) that we proposed in Section 4. The analysis derives the optimum throughput and area of processing systems composed of VLSI chips for the two-dimensional FHP lattice gas problem. We define the design parameters for each system and derive the design curves and optimum values of those parameters. For the analysis, we assume that a memory system capable of providing full bandwidth to the processor system is available.* Finally, we compare the systems on the basis of maximum throughput, total system area, and throughput-to-area ratio. We also discuss the relative advantages and disadvantages of both architectures with an emphasis on system complexity and ease of implementation.

2.7. Wide-Serial Architecture (WSA)

The wide-serial architecture (**WSA**) has system parameters: (assumes 1 pipeline stage per chip, P processing elements wide)

$$N = k \ \ chips \qquad \text{(System Area) (2-1)}$$

$$R = F \cdot P \cdot k \ \ \frac{sites}{second} \qquad \text{(System Throughput) (2-2)}$$

and chip constraints

$$2D \cdot P \le \Pi \qquad \text{(Chip Pins) (2-3)}$$

$$\beta(2L+7P+3) + \gamma P \le \alpha \qquad \text{(Chip Area) (2-4)}$$

where

*This is a very important assumption.

N is the total number of chips constituting the processor,

P is the number of PE's per VLSI chip,

k is the total depth in PE's of the processor pipeline (path length),

F is the major cycle time of the chip,

D is the number of bits required to represent the state of a lattice site,

L is the number of sites along an edge of the square lattice,

Π is the total number of pins usable for input/output,

β is the area of a shift register that holds a site value, in λ^2,

γ is the area of a PE, in λ^2,

α is the total usable chip area, in λ^2.

For convenience, we also define:

$$B = \frac{\beta}{\alpha} = \text{normalized site storage area}$$

$$\Gamma = \frac{\gamma}{\alpha} = \text{normalized processor area}$$

Less formally, this says that the number of chips that we need for the processor equals the total pipeline depth required, k. The processing rate that this system achieves is equal to the depth of the pipeline, multiplied by the number of processors at each depth, multiplied by the rate at which a processor computes new sites. We are assuming that each VLSI chip will contain only a single wide parallel pipeline stage. That is, the chip is not internally pipelined with wide-serial processors.

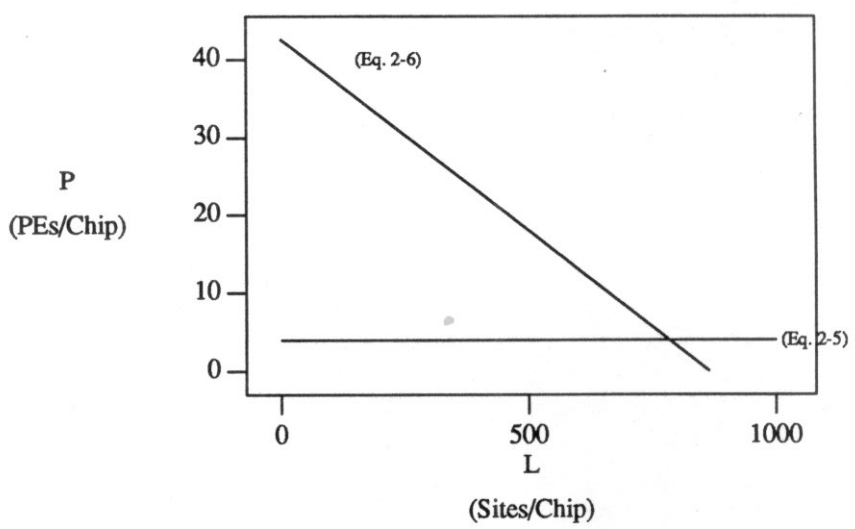
We wish to maximize R subject to having a fixed number of chips, $N = N_0$, and subject to constraints on the pin count and area of the VLSI custom chip. Notice that the problem is equivalent to maximizing P subject to the chip constraints because $R = F \cdot P \cdot k = F \cdot P \cdot N$, where F and N are fixed (N is fixed at N_0).

The constraints are described in the L–P plane by the following two inequalities:

$$P \le \frac{\Pi}{2D} \tag{2-5}$$

$$P \le \frac{1-3B-2BL}{7B+\Gamma} \tag{2-6}$$

If we consider an example where $D = 8$, $\Pi = 72$, $B = 576\times10^{-6}$, and $\Gamma = 19.4\times10^{-3}$ (figures derived from our actual layouts) we get the following graph:



The chip constraints require that the operating point determined by P and L lie below both curves. The intersection of the two curves is $P \approx 4$ and $L \approx 785$. Beyond that point, we need to decrease the number of processors on a chip to make room for more memory — an undesirable situation because throughput then drops off linearly. Furthermore, we want L to be as big as possible, so the corner is the logical choice of operating point.

We are also interested in the ultimate maximum performance that the architecture can deliver using any number of chips. It is easy to see that the maximum throughput for a fixed clock frequency, F, comes when the pipeline depth, k, is at a maximum. A maximum value, $k_{max} = L$, arises because at that point the pipeline contains all the values of the sites in the lattice and there is no new data to introduce into the processor pipeline. The maximum values for processor system area and processor system throughput are therefore:

$$N_{max} = L \quad chips \tag{2-7}$$

$$R_{max} = \frac{\Pi}{2D} \cdot F \cdot L \quad \frac{sites}{sec} \tag{2-8}$$

It is also interesting to note that there is an upper bound on L even if we were to accept arbitrarily slow computation. At a certain point all the chip area would be used for memory, leaving no room for PE's.

The major limitation of this architecture is that the largest problem instance is fixed by the chip technology, but it has the redeeming features of simplicity, ease of implementation, and small main memory bandwidth.

2.8. Sternberg Partitioned Architecture (SPA)

This processor computes updates for a lattice L sites on a side by partitioning the lattice into non-overlapping slices that are each W sites wide (there are $\frac{L}{W}$ such slices). Each of the VLSI chips that compose the processor computes P_w slices and the computation of each slice is pipelined on the chip to a depth P_k (see Figure 2-4). It is then easy to see that the system has area and throughput:

$$N = \frac{\frac{L}{W}}{P_w} \cdot \frac{k}{P_k} \quad chips \qquad \text{(System Area) (2-9)}$$

$$R = F \cdot k \cdot \frac{L}{W} \quad \frac{sites}{sec} \qquad \text{(System Throughput) (2-10)}$$

To derive the constraints on the VLSI chip, notice that the communication path between chips in the direction of the data pipeline requires $2DP_w$ pins, and that the "slice to slice" path requires $2EP_k$, where E is the number of bits required to complete the information contained in a single site's neighborhood, when that neighborhood is split across a slice boundary. However, the chip must use no more than α area, of which processors each require γ, and memory to hold a site value requires β. Thus the whole chip is governed by the constraints

$$2DP_w + 2EP_k \le \Pi \qquad \text{(Chip Pins) (2-11)}$$

$$((2W+9)\beta + \gamma)P_w P_k \le \alpha \qquad \text{(Chip Area) (2-12)}$$

We again wish to maximize throughput with respect to a fixed number of chips, $N = N_0$, while at the same time satisfying the VLSI chip constraints of area and bandwidth. This again turns out to be equivalent to maximizing the total number of processors on the chip because we can easily verify by direct substitution that $R = F \cdot k \cdot \frac{L}{W} = P_w P_k \cdot F \cdot N_0$. Since F and N_0 are fixed, it suffices to maximize the product $P_w P_k = P$ subject to the constraints above.

To evaluate the design space of **SPA**, it is helpful to view it in the W–P plane. We do this via a change of variables:

$$P = P_w P_k .$$

Rewriting the chip inequalities yields:

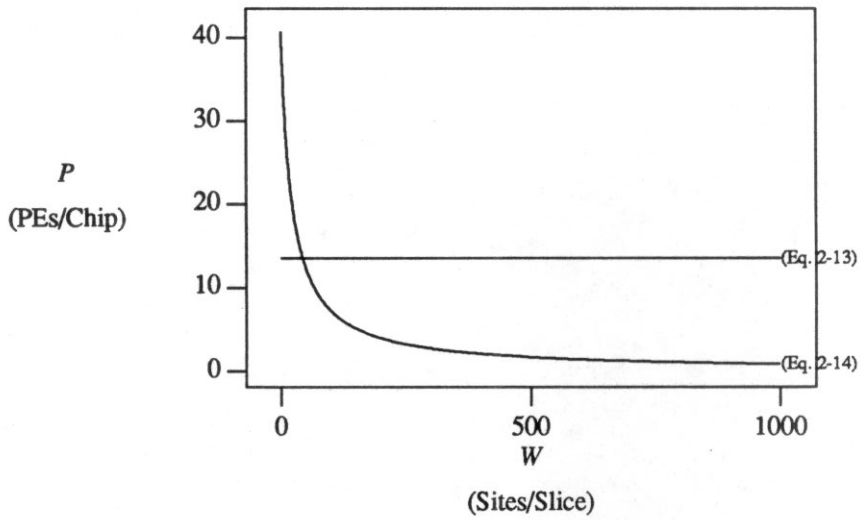
$$2DP_w + 2E\frac{P}{P_w} \le \Pi \qquad (2\text{-}13)$$

$$((2W+9)B + \Gamma)P \le 1 \qquad (2\text{-}14)$$

where P_w, P, and W are variables. This is the logical choice of variables for this architecture because they are the ones that are constrained by the chip technology and govern the optimal design of the chip. Once we know good values for them, a machine which can compute for an arbitrary lattice

width L can be built by increasing the number of slices of width W.

When these curves are projected onto the $W-P$ plane using the values for D, Π, B from the previous example; and setting E to 3 (three bits must be passed to complete an LGCA neighborhood) we have:



The constant curve is a projection of the first constraint where P_w is given the value which permits P to achieve its maximum value. For this example, this occurs at $P_w = \frac{9}{4}$. As before, we need to operate below both curves, and the corner at $P \approx 13.5$ and $W \approx 43$ yields the best choice. Beyond this point, throughput drops off quite rapidly as the silicon real estate is used by memory.

2.9. Discussion

The above analysis gives us two different viewpoints from which to make a comparison between these two architectures. Ignoring extensibility, we can make a comparison between the two designs when they are optimized for throughput, as they were in the preceding analysis. To include extensibility as a criterion, we can make the comparison by using a slight variant of **WSA** which allows for extensibility by sacrificing processing speed.

First, let us compare the designs optimized for throughput without regard to extensibility. The optimal **WSA** configuration limits the lattice length to $L = 785$. Both systems, **WSA** and **SPA**, have throughput rates which grow linearly with the number of chips. However, **SPA** is three times faster than **WSA** (**SPA** has twelve processors per chip while **WSA** has four). On the other hand, the **SPA** system requires four times as much main memory bandwidth as the **WSA** system: 262 bits/tick versus 64 bits/tick.

The above argument contains a bias in favor of **SPA**. System timing is an important consideration which can make it difficult to clock **SPA** as fast as **WSA**. The **WSA** architecture has connectivity

only in one dimension whereas the **SPA** system requires communication in both the pipeline direction and the synchronous side-to-side data paths. This added complexity is a more pronounced drawback for **SPA** when extensibility is considered, as we will mention below. The conclusion in both cases favors the **WSA** system when it comes to considering an implementation. There is also the matter of the data access pattern in the memory. The **WSA** machine accesses the data in a strict raster scan pattern which is simpler than the row-staggered pattern that the **SPA** scheme requires for its operation.

The **SPA** architecture has one considerable advantage over the **WSA** scheme: extensibility. Smaller instances of an **SPA** machine can be joined together to form a machine that computes a larger lattice. This is not true for the **WSA** case, where computation is limited to lattice sizes which do not exceed L as given by the chip area constraint, because all the required data must fit on the chip. This requirement is relaxed in the **SPA** scheme because data can be moved between adjacent chips as W is adjusted to the chip constraints and an arbitrary lattice width L can be supported by composing a suitable number of slices. In this respect the two schemes seem incomparable.

Our second point of view on the comparison of these two architectures is facilitated by considering a slight variation of **WSA** which allows extension of the lattice size. The extension can be accomplished by moving a portion of the shift register off chip. The pin constraints given previously, with the same constants, allow only one processor per chip in this case. A stage in the pipeline consists of a processor chip and associated shift registers sufficient to hold the remainder of the $2L + 10$ node values which do not fit onto the processor chip. We will call this version of **WSA** **WSA-E**.

Both systems, **SPA** and **WSA-E**, have throughput rates that grow linearly with the number of chips in the system for a fixed lattice size L. However, the constant of proportionality between the two rates grows with increasing lattice size. The reason is that the number of processors per unit chip area is independent of lattice size for **SPA**, whereas it decreases with increasing lattice size for **WSA-E**. So, for instance, given the same number of chips and a lattice size $L \leq 785$, the **SPA** system is twelve times faster than **WSA-E** because it has twelve processors per chip as opposed to one per chip.

A better understanding of the contrasts between the two systems can be obtained by looking at requirements for main memory bandwidth and storage area per processor. **WSA-E** has a constant bandwidth requirement of 16 bits per clock tick and requires $(2L + 10)B$ storage area per processor; **SPA** has a main memory bandwidth requirement of $\frac{L}{3}$ bits per tick and requires $(1284)B$ area per processor. For a fixed processing rate the penalty for larger lattice size is either linear growth in the number of chips for the **WSA-E** system, or linear growth in the main memory bandwidth in the **SPA** case. For example, if $L = 1000$, then **WSA-E** requires about twice as much area as **SPA**, while requiring about one twentieth as much bandwidth.

2.10. Summary

The technique described in this chapter can be used to analyze other architectures. It offers a systematic way to arrive at important system parameters. Its greatest limitation is that it depends on constants of proportionality and offers no meaningful asymptotic information. It also requires that the design has progressed at least as far as the layout of certain basic cells so that areas can be used in calculations.

Another limitation is that the technique neglects interconnection and wiring area, which we later found to have significant effect. It reduced the number of PEs from 4 to 2 and the line length, L, from 785 to 512 in our actual implementation (see Chapter 4).

The assumption that the required memory bandwidth is always available from the host is not realistic. Initially, we had included a parameter which characterized the host/PE bandwidth mismatch. However, the algebra obscured the analysis and comparison, and we therefore chose to present only the simplified presentation.

Nevertheless, this analysis provided enough information to help us choose our implementation and rule out alternate architectures. Given the simplicity and directness of **WSA**, and given that performance could be increased almost arbitrarily by cascading chips, we decided to implement the **WSA** architecture.

Chapter 3
A Probabilistic Model For Clock Skew

3.1. Introduction

The accumulation of clock skew, the differences in arrival times of signals in a computing system with a central clock, is one of the factors that limit the speed of such systems. While some researchers have attempted to describe the underlying physical causes of skew [40,41], there is little published material describing models of skew accumulation and the asymptotic behavior of skew with increasing system size.

Fisher and Kung in [27] present two models and derive bounds on clock skew under each. They assume, in each case, that the global clock is distributed via a rooted binary tree of buffers and wires.

The first of their two models, the "difference model," specifies that the skew between two clocked nodes is proportional to the difference in the path lengths from each clocked node to their nearest common ancestor in the distribution tree. "Path length" means the actual geometric length, not the number of edge traversals in a representative graph. Under this model, both one and two-dimensional arrays can be clocked without skew by distributing the global signal via a topology that has equal path length to each clocked node. An H-tree is just one example of such a topology.

The second model, the "summation model," is less optimistic. It states that the skew between two clocked nodes is proportional to the sum of the path lengths from each node to their nearest common ancestor. Using this model, Fisher and Kung were able to show that one-dimensional arrays of processors can be clocked with a constant amount of skew, whereas two-dimensional arrays cannot. They establish a lower bound of $\Omega(\sqrt{N})$, and this model leads to a skew of $\Theta(N)$ for an H-tree.

Both of these models ignore what we consider to be a fundamental property of skew, namely, its roots in the random variations of propagation time through buffers and wires.

Others [42] have developed equations that relate skew to system performance and stability, in terms of simple timing parameters such as the propagation time and the settling time of the components of the processing elements. The equations specify bounds on the skew for proper operation in

terms of other system timing parameters. No attempt is made to address the nature of the origin of skew, nor to relate its rate of growth with system size.

This chapter presents and analyzes a probabilistic model for the accumulation of skew in a globally distributed signal. We determine upper bounds for the expected clock skew between processing elements in a processor array, and show that under the assumption that the delay is the same at each stage of the clock distribution tree, any array can be clocked with expected skew that is $O(\log N)$ where N is the number of processing elements.

The organization of the chapter is as follows. In Section 3-2 we present our formal model of global signal distribution and skew accumulation. Section 3-3 presents an analysis of the model and derives upper bounds for the expected skew in a global signal. Section 3-4 applies the results to two examples, and we conclude by comparing our results to previously published results and by discussing the implications of these upper bounds on the construction of large synchronous systems.

3.2. A General Model of Signal Distribution

A global signal, such as a clock, is distributed throughout a processing system by a signal distribution system. The distribution system is composed of a number of buffers (amplifiers) and wires which may be organized in a number of different ways. Two common structures are a bus and a tree.

The clock distribution system can be represented by a graph. It has a single distinguished vertex which is called the *source*. This is where the origin of the global signal is located, and it is the only input to the distribution system. The distribution system can have multiple destinations, but for practical reasons, there is exactly one path from the source to each destination. We assume that a destination is a processing element (PE) which may have its own internal signal distribution system. Using the tools which are developed below, the internal system can be modeled in a similar manner to the global signal distribution system. For our purposes, however, we can think of it as hidden; we concern ourselves only with the global signal system.

There is a second graph superimposed on the clock distribution graph. It is a *communication graph* which specifies the PE-to-PE connectivity. It can take any topology: linear list, mesh, complete, bipartite, etc, and forms connections only among the PEs. The PEs use the global clock to synchronize their communication along the communication graph.

Each buffer and wire in the clock distribution system propagates and delays its incoming signal. Therefore, it is natural to associate every delay element in the signal distribution system, whether a buffer or a wire, with a real random variable, d_j. The value of the random variable gives the delay contribution of that element. The delay at any stage of the clock distribution system is the sum of the delays from the signal source to that point. An equivalent process takes place in the model. The delay at any point is a real random variable which is the sum of all the random variables along the path from the source to that point.

These definitions constitute the essence of the model. They make it very simple, but extremely general, and allow one to model any clock distribution system. Geometry can be incorporated into the model by attaching an appropriate probability density function to wire delays. There is also the freedom to analyze as much or as little as desired by creating simplified models, in which buffer delays or wire delays can be ignored entirely.

Our primary interest is skew, the distribution of arrival times of a particular clock pulse to all of the PEs that communicate. Since the model is probabilistic, it is not possible to give an expression for the worst-case skew. Rather, we derive an expression for the expected maximum skew by assuming that the PEs are interconnected with a complete graph.

So far, we have made no mention of any particular probability density functions. The total delay through the distribution system, i.e., the arrival time of a clock pulse to a PE, is the sum of a number of random variables. In many cases, it quickly converges to a normal distribution, by the Central Limit Theorem [43-45]. Thus, in the case of the skew computations, the actual distributions attached to buffers and wires are usually relatively unimportant.

The arrival times of a signal to the N PEs constitute a random sample of size N. From this sample, find the difference between the largest of them, A_{max}, and smallest, A_{min}. The random variable $R = A_{max} - A_{min}$ is called the *range* of the sample. The range is equivalent to the skew in the signal distribution system. The maximum clocking frequency clearly can be no faster than $1/R$.

3.3. Analysis and Upper Bounds

The literature contains techniques to compute the expected range of a set of independent identically distributed (iid) random variables. However, little is described about the case when the variables are dependent, as they are in the clock tree. Fortunately, it is possible to use the statistics of iid variates as an upper bound on the statistics of the dependent variates of the clock tree. The relationship is given by Theorem 3-1, which can be paraphrased in the following informal way:

> The expected range of a set of random variables, which are dependent because they are the sums of overlapping variables, is no greater than the expected range of the corresponding set of independent random variables.

Let y_i, $i = 1, 2, ..., N$ be independent identically distributed (iid) real random variables, with $N = kn$, and let the sets σ_j, $j = 1, 2, ..., n$ be n disjoint subsets of distinct y_i, each of cardinality k. Let the τ_i be similarly defined, with k distinct elements each, except that they are not necessarily pairwise disjoint. Define the corresponding sums of the y_i by

$$s_j = \sum_{y \in \sigma_j} y$$

and

$$t_j = \sum_{y \in \tau_j} y$$

We want to show that the expected range of the s_j dominates the expected range of the t_j, and so any upper bound for the former also holds for the latter. First we need two lemmas.

Lemma 3-1. Let $F_A(x)$ and $F_B(x)$ be probability distribution functions for random variables A and B respectively, and suppose further that F_A and F_B are differentiable and A and B have finite means and variances. If $F_A(x) \geq F_B(x)$ for all x, then $\mathrm{E}(A) \leq \mathrm{E}(B)$.

Proof. Integrating $x\, dF(x)$ by parts from a to b for F_B and F_A and subtracting, we get

$$E(B) - E(A) = \lim_{\substack{a \to -\infty \\ b \to +\infty}} \left[b\Delta F(b) - a\Delta F(a) - \int_a^b \Delta F(x)\, dx \right]$$

where $\Delta F = F_B - F_A \leq 0$. To show that the first two terms go to zero as a and b approach $-\infty$ and $+\infty$ respectively, we apply l'Hospital's Rule:

$$\frac{\dfrac{d}{dq}\left[\Delta F(q)\right]}{\dfrac{d}{dq}\left[\dfrac{1}{q}\right]} = \frac{\Delta F'(q)}{-\dfrac{1}{q^2}} = -q^2 \Delta F'(q)$$

F_A and F_B are differentiable and have finite variances, so the integral

$$\int_{-\infty}^{+\infty} x^2\, \Delta F'(x)\, dx \tag{3-1}$$

is finite. This implies that the integrand of (3-1) must go to zero as x approaches $+\infty$ and $-\infty$. Therefore

$$E(B) - E(A) = -\int_{-\infty}^{+\infty} \Delta F(x)\, dx \geq 0$$

□

The next lemma expresses the intuition that the pre-condition $y \leq \beta$ can only make $y \leq \alpha$ more probable, no matter what α and β are.

Lemma 3-2. For any α and β, and any continuous probability density P,

$$\mathrm{P}(y \leq \alpha \mid y \leq \beta) \geq \mathrm{P}(y \leq \alpha)$$

Proof. For convenience write

$$P(y \le \alpha) = G(\alpha)$$

$$P(y \le \alpha \mid y \le \beta) = G(\alpha \mid \beta)$$

$$P(y \le \alpha, y \le \beta) = G(\alpha, \beta)$$

so that we want to show

$$G(\alpha \mid \beta) = \frac{G(\alpha, \beta)}{G(\beta)} \ge G(\alpha)$$

First, consider the case $\alpha < \beta$. Since

$$G(\alpha, \beta) = G(\min[\alpha, \beta])$$

we have

$$G(\alpha \mid \beta) = \frac{G(\alpha)}{G(\beta)} \ge G(\alpha)$$

When $\alpha > \beta$ we have simply

$$G(\alpha \mid \beta) = \frac{G(\beta)}{G(\beta)} = 1 \ge G(\alpha)$$

□

Lemma 3-2 is easily generalized for any number of conditions to

$$G(\alpha \mid \beta, \gamma, \cdots) \ge G(\alpha) \tag{3-2}$$

and in fact to the probability distribution of sums of variables, conditioned on other sums, so long as the inequalities in the conditions all go the same way. That is, write any $\sum y_i \le x$ as $y_k \le x - \sum_{i \ne k} y_i$ for some k and apply Lemma 3-2. There are no restrictions on α and β.

Theorem 3-1. The expected range of the s_j is no smaller than the expected range of the t_j.

Proof. Note first that because E[range] = E[max − min] = E[max] − E[min], it suffices to show that the expected max of the s_j cannot be less than the expected max of the t_j; the appropriate inequality for the min follows by a symmetric argument.

Let S and T be the maximum of the sums s_j, $j = 1, 2, ..., n$ and t_j, $j = 1, 2, ..., n$ respectively, and F_S and F_T their distribution functions. By Lemma 3-1, it suffices to show that $F_T(x) \ge F_S(x)$ for every x.

To simplify notation, let S_i denote the event $s_i = \sum_{y \in \sigma_i} y \le x$ and T_i the event $t_i = \sum_{y \in \tau_i} y \le x$. The distribution function $F_S(x)$ can then be written as

$$F_S(x) = \mathrm{P}(S_1, S_2, \ldots, S_n)$$

$$= \mathrm{P}(S_1)\,\mathrm{P}(S_2)\cdots\mathrm{P}(S_n) \text{ because the } S_i \text{ are independent.}$$

The distribution of T is

$$F_T(x) = \mathrm{P}(T_1, T_2, \ldots, T_n)$$

$$= \mathrm{P}(T_1)\mathrm{P}(T_2 \mid T_1)\mathrm{P}(T_3 \mid T_1, T_2)\mathrm{P}(T_4 \mid T_1, T_2, T_3),\ldots$$

by iterated Bayes' rule. Each factor in this product is

$$\mathrm{P}(t_i \le x \mid t_1 \le x,\ t_2 \le x,\ \cdots,\ t_{i-1} \le x)$$

All the conditioning is of the form $\Sigma y \le x$ and so we can apply the generalized version of Lemma 3-2 to show that each factor is $\ge \mathrm{P}(T_i)$. The result $F_T \ge F_S$ follows from the fact that the s_i and t_i are identically distributed, each being the sum of k iid variables. □

Now that we have established this theorem, any bound that we show for iid variates is an upper bound for the variates that arise in the clock distribution system model.

At this point we are going to assume that the arrival times are Gaussian, motivated by the Central Limit Theorem. Although no closed-form expression is known for the expected value and the variance of the range of N iid Gaussian distributed random variables, it is possible to obtain asymptotic expressions. We present these as Theorem 3-2, the proof of which is due to Cramér.

Theorem 3-2. [45] Let x_i, $i = 1,\ldots,N$ be random samples from an $n(\mu,\sigma^2)$ distribution,* and let $R = x_{\max} - x_{\min}$ be the difference of the largest and smallest x_i. Then the expected value of R is asymptotically:

$$\mathrm{E[R]} = \sigma\left[\frac{4\log N - \log\log N - \log 4\pi + 2C}{(2\log N)^{1/2}} + O\left[\frac{1}{\log N}\right]\right] \tag{3-3}$$

where $C \approx 0.5772...$ is Euler's constant. The variance of R is given by

$$\mathrm{Var[R]} = \frac{\sigma^2}{\log N}\frac{\pi^2}{6} + O\left[\frac{1}{\log^2 N}\right] \tag{3-4}$$

Equation (3-3) is therefore an asymptotic upper bound on the expected skew in a clock distribution tree with N leaves. Note that σ will in general be a function of N.

*We use the notation $n(\mu,\sigma^2)$ to denote a normal distribution with mean μ and variance σ^2.

3.4. Examples

We now analyze two examples of global signal distribution systems. This is meant to clarify how a model is developed for a real problem. The examples represent what we consider to be common, typical clock distribution systems, but they are not intended to represent the full scope of all possibilities.

3.5. Metric-Free Tree

The first example is a *metric-free* tree. This type of topology could be used to implement a large-scale distribution system which would provide a clock to chips on a board or to boards in a system. It does not constrain the circuit to be planar, so it is possible to equalize the lengths of all wires in the tree. Therefore, every wire has the same probability distribution for delay, which can be lumped with the delay of the buffer that follows it. This results in a model of a tree of buffers without wires.

Assume that the global clock signal is distributed via a binary tree of buffers and wires. The root of the tree is the source of the signal, the PEs are placed at the leaves, and the intervening levels consist of buffers and wires. See Figure 3-1 for a schematic representation of the tree.

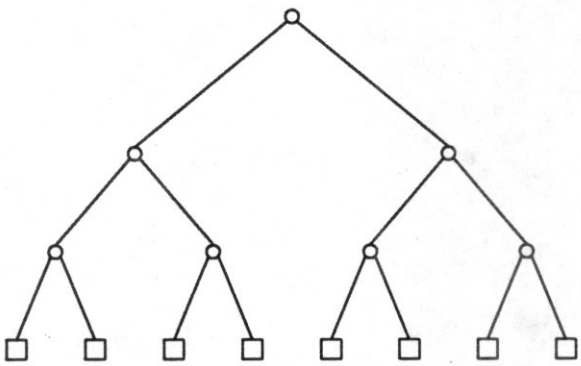


Figure 3-1: Clock Distribution Tree

In the figure, internal nodes, represented by small circles, are buffers which retransmit the clock signal. The leaf nodes of the tree, represented by squares, are the PEs that perform the actual computation and that communicate among themselves. Lines connecting nodes of the clock tree represent wires that conduct the clock signal to all the PEs.

Let the delay through a buffer of the clock distribution tree be a real random variable, d_i. The arrival time of a clock signal to any PE is the sum of the delays along the path from the root of the tree

to the PE. Remember that all the delay is caused by the buffers; the effect of a wire is absorbed by the effect of the buffer that follows it. The arrival time at leaf i, A_i, is therefore the random variable

$$A_i = \sum d_j \quad \textit{where the } d_j \textit{ lie on the path from the root to } i.$$

In order to apply Theorem 3-2, we must estimate the underlying distribution of the A_i.

Assuming that there are N PEs, each A_i is the sum of $\log N$ d_j's, and that each d_j has variance σ_b^2. By our Gaussian assumption, the A_i have the distribution $n(\mu_b \log N, \sigma_b^2 \log N)$. Applying Theorem 3-2 with this distribution, we find that the expected skew is

$$\mathrm{E[Skew]} = \sigma_b \sqrt{\log N}\left[\frac{4\log N - \log\log N - \log 4\pi + 2C}{(2\log N)^{1/2}} + O\left[\frac{1}{\log N}\right]\right] \tag{3-5}$$

$$= \sigma_b \frac{4}{\sqrt{2}} \log N + \text{lower order terms}$$

$$= \Theta(\log N)$$

The variance of the skew is

$$\mathrm{Var[skew]} = \sigma_b^2 \frac{\pi^2}{6} + O\left[\frac{1}{\log^2 N}\right]$$

which goes to a constant as $N \to \infty$.

Computer simulations corroborate the asymptotic skew results. Figure 2 shows the asymptotic curve, Equation (3), along with data gathered from simulation of the range of arrival times in a binary clock tree and the range of N iid $n(0, \log N)$ random variables. Both simulations present the average behavior from 100 trials. The simulations confirm that the range of the iid variates is an upper bound for the range of the dependent variables of the clock distribution tree, as predicted by Theorem 3-1.

To check the result when the component delays are not Gaussian, we simulated a clock tree where the buffer delays were uniform over [-.5,+.5]. These data were compared to the results from a similar computation when the buffer delay was normally distributed with mean 0 and variance 1/12 (The sum of k uniform [-.5,+.5] variates converges to $n(0, k/12)$). Figure 3-3 presents the results. Even for trees of small depth, the expected range in both cases is nearly identical. This is due to the rapid convergence of the sums of random variables to a normal distribution.

The explicit inclusion of wire delays into the model does not significantly alter the results. Wires can be considered to contribute an additional random delay at each level of the tree. Assuming that wire delays are distributed similarly to the buffer delays, the effect is to increase the variance of the distribution of arrival times by a constant factor. This does not alter the asymptotic behavior of skew.

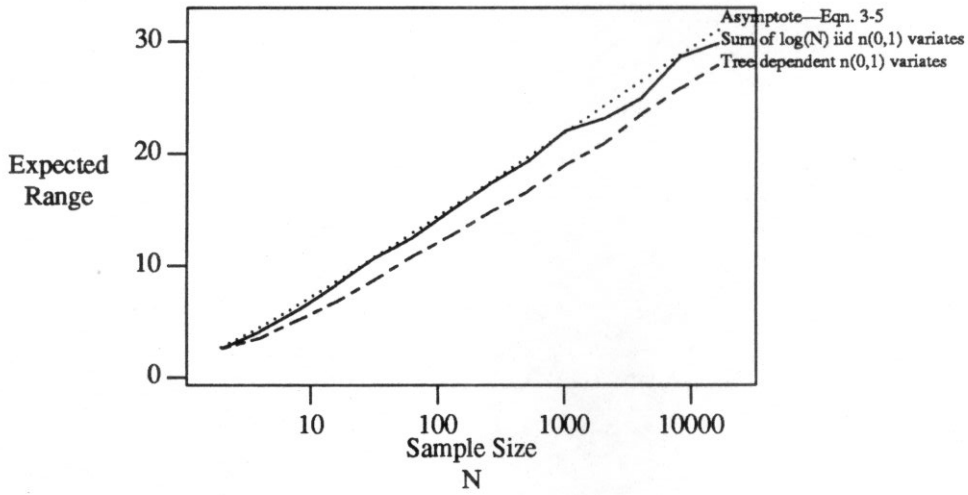


Figure 3-2: Range / Skew of Random Variables (Metric Free)

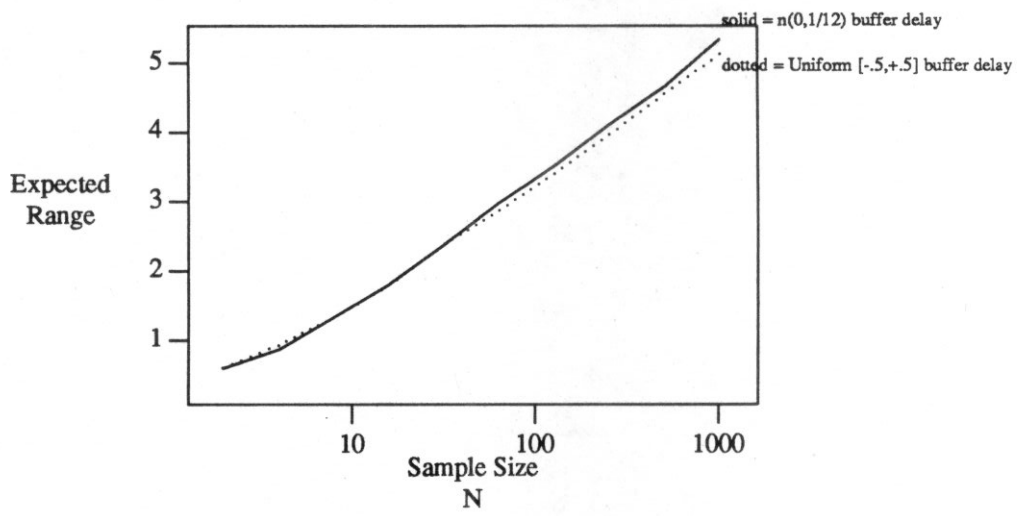


Figure 3-3: Skew in Tree with Uniform and Gaussian Delay

3.6. Metric Tree

The second example is a *metric* tree. This is the type of system that is discussed in [27] and is typical of systems described for VLSI. The central assumption of this topology is that the circuit must be embedded in the plane. If the embedding is to be area-efficient [46], then the wires that connect buffers cannot be the same length everywhere. The delay through a wire therefore depends on the its location in the tree, and cannot be lumped with a buffer delay.

A common tree of this type is the H-tree [47]. We choose to model it here because it is well known, it is feasible to implement in VLSI, and it the focus of the analysis in [27] so we may compare our results with theirs.

There are two distinct views of the effects of increasing system size (number of PEs) under the metric assumption. The first is to assume that a tree with an arbitrary number of leaves can be embedded in the fixed area of the integrated circuit. The alternative to this view sets a lower limit on the size of the smallest feature; in this case the size of a wire at the tree's leaves. Each preceding level is progressively larger and the area of the entire clock tree grows with increasing system size. This view ignores the effect of shrinking feature size but is compatible with increases in chip die size. We will adopt the second model, as did Fisher and Kung.

Refer again to Figure 3-1, keeping in mind that in the metric case the wires are not all of the same length. Assume that every buffer delay, d_j, is $n(\mu_b,\sigma_b^2)$. For this analysis, we will also assume that a wire delay, w_j, is Gaussian distributed with a mean value and a variance proportional to its length. The linear relationship for the variance can be justified by considering a long wire to be equivalent to two shorter wires placed end to end. The propagation delay of the long wire is equal to the sum of the propagation delays of the two shorter wires. The expected values add, as do the variances because the delays of the short wires are independent.

The wire delay at the leaves of the tree is $n(\mu_w,\sigma_w^2)$ distributed. Because wire length doubles at each higher level of the tree, the distribution of w_j can be written as a function of the depth, d, of the wire. We find that w_j is $n(\mu_w\frac{N}{2^d}, \sigma_w^2\frac{N}{2^d})$ where $1 \le d \le \log N$.

The total delay, A_i, is the sum of $\log N$ buffer delays and the sum of wire delays from each level of the tree. The wire delays form the geometric series $(1+2+4+\cdots+\frac{N}{2}) = N-1$. The total delay, A_i, therefore has the distribution $n(\mu_b \log N+\mu_w(N-1), \sigma_b^2 \log N+\sigma_w^2(N-1))$. As $N\to\infty$, the linear (wire) term dominates. The expected skew is therefore

$$\mathrm{E[skew]} = \sigma_w\sqrt{N}\left[\frac{4\log N-\log\log N-\log 4\pi+2C}{\sqrt{2\log N}} + O\left[\frac{1}{\log N}\right]\right] \tag{3-6}$$

$$= \sigma_w\frac{4}{\sqrt{2}}\sqrt{N\log N} + \text{lower order terms}$$

$$= \Theta(\sqrt{N \log N})$$

and the variance is given by

$$\text{Var[skew]} = \frac{\sigma_w^2 N}{\log N}\left[\frac{\pi^2}{6}\right] + O\left[\frac{1}{\log N}\right]$$

Figure 3-4 shows the results of computer simulations. The dotted curve is the asymptotic formula, Equation (3-6), the solid curve is the range of iid variates, and the dashed curve shows the range of tree-dependent variates. The simulated cases represent the average behavior from 100 trials. The discrepancy is greater in this case because the shared variates, representing deviation from the independence assumption, are near the root of the tree, where the wire lengths are longer. Thus, this bound is not very tight, and a more detailed analysis might improve it.

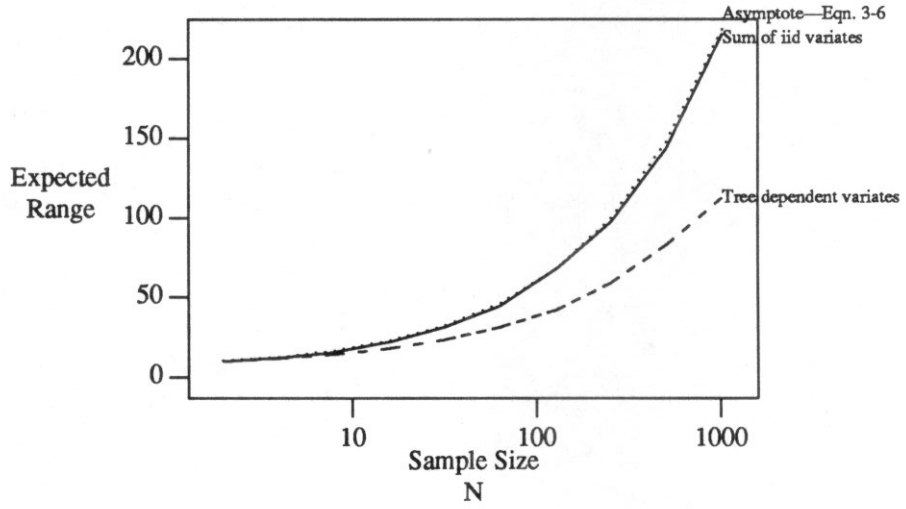


Figure 3-4: Range / Skew of Random Variables (Metric)

3.7. Discussion

It is now possible to give an estimate on the probability that the sample value of the skew is outside a certain range. Assume that X is a random variable with mean μ and variance σ^2. Then the one-sided Chebyshev inequality [43] yields an upper bound on the probability of exceeding the mean skew by an amount a:

$$P(X > (\mu+a)) \leq \frac{\sigma^2}{\sigma^2+a^2}$$

Now let $a = \alpha\mu$; that is, α is the fractional deviation from the expected value of skew. Then, using our estimate, in both the metric and the metric-free case, we have an estimate of an upper bound as $N \to \infty$:

$$P(X > (\mu+a)) \leq \frac{\frac{\pi^2}{6}}{\frac{\pi^2}{6} + 8(\log N)^2\alpha^2}$$

3.8. Conclusions

As long as VLSI offers a finite resource, large systems must be constructed from many chips. The clock distribution system can then be non-planar, since it is not restricted to a single VLSI integrated circuit. We therefore consider our metric-free results to be important in answering the question of the ultimate limit of synchronous computation. Our finding that skew grows as the logarithm of the system size encourages us to believe that very large synchronous systems are feasible. It appears that this estimate is new.

It is informative to compare our results for the metric tree with those of Fisher and Kung. Their summation model yielded a lower bound of $\Omega(\sqrt{N})$ for the skew in a processor array with N PEs. The worst case performance of the H-tree under their summation model is $\Theta(N)$, whereas we show that expected skew is only $\Theta(\sqrt{N \log N})$. This places our result between their summation model bound and their graph-theoretic lower bound.

3.9. Acknowledgements

We wish to thank J.P. Singh for helpful discussions and literature survey [48].

Chapter 4
Lattice Gas Cellular Automaton CMOS Chip

4.1. Introduction

This chapter describes the design and implementation of the **L**attice **G**as **C**ellular **A**utomaton (**LGCA**) special purpose chip. It is a 64 pin DIP fabricated by MOSIS in 3μ p-well CMOS and performs two-dimensional lattice gas updates according to the FHP lattice gas model.

Since it is a pipelined processor, the chips can be concatenated for increased performance. Moreover, the total performance is linear in the number of chips. Chips can be added to deepen the pipeline without any special programming of the hardware, although the software must compensate for the increased pipeline latency.

The LGCA chip is an implementation of the **WSA** architecture described in Chapter 2. It accepts a serial stream of site values and emits a stream of values in the same sequence after one generation of computation. This allows the chips to be used in a pipelined fashion because every chip in the pipeline receives a stream of input data values in the same temporal sequence. Every chip and every position is equivalent from the standpoint of the data flow and the computation. Therefore, no position-dependent hardware is required on the chip.

The processing takes place on diagonal wavefront through space and time as illustrated by Figure 4-1.

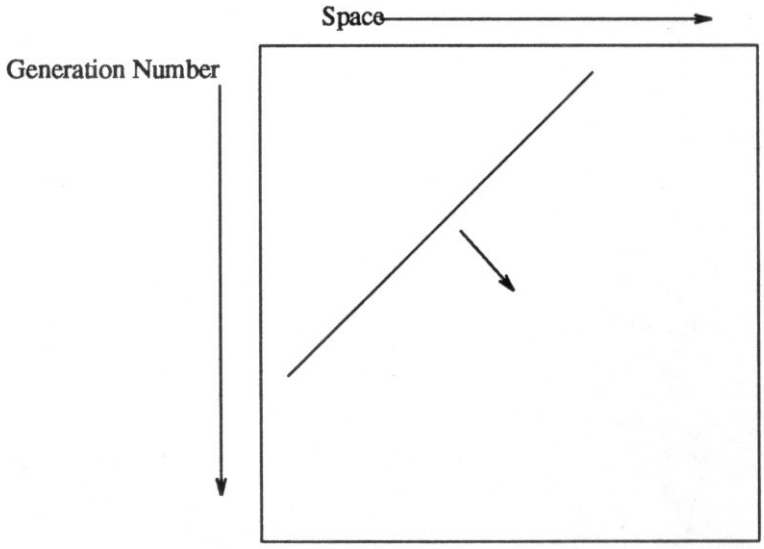


Figure 4-1: Space/Time Diagram of Pipelined Computation

In designing the chip, we undertook to satisfy the following goals:

- Efficient use of VLSI resources.
- Short design cycle.
- Ease of verification and testing.
- Simple integration with the host.
- High performance on the FHP problem.
- Some degree of programmability.

4.2. The Lattice and Data Encoding

The LGCA chip performs a particle simulation of a two-dimensional fluid according to the FHP model. The fluid is modeled by a hexagonally connected lattice with unit-length edges. Particles with unit mass and unit velocity move around in the lattice and collide at sites which are located at the intersection of the edges. See Chapter 1 for details of the FHP model. In theory, the lattice can be finite or infinite in any direction, but the chip can only support simulations in which one dimension is fixed. In this implementation, the lattice is cut on a zig-zag along both sides to make it as rectilinear as possible. There is no limit on the number of rows in the lattice, but a top row is designated. Thus the lattice can be considered semi-infinite in one dimension.

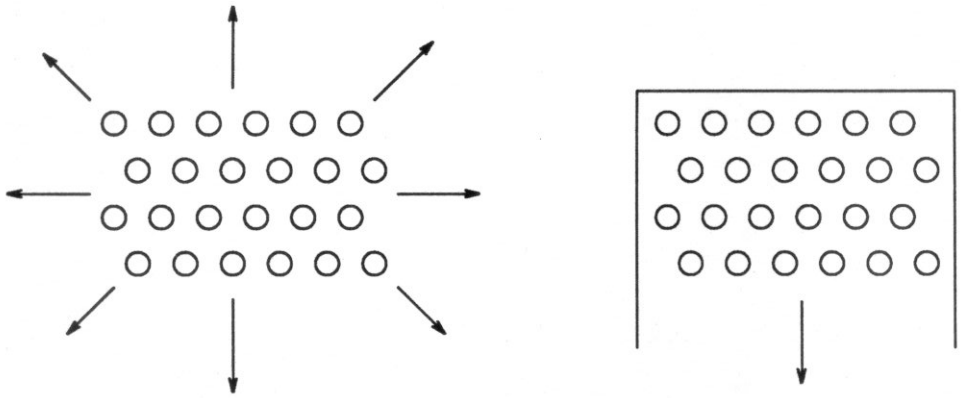


Figure 4-2: Theoretical and Implemented Lattices

Data for the fluid simulation are kept in an array, which is scanned to produce a serial stream of lattice site values for input to the processor pipeline. The value stored at a site is an encoding of the number of particles at that lattice site and their directions. We define the state of a lattice site to be the state of its *outgoing* links, i.e., which links have outgoing particles.

The converse definition, that the state of a site is described by the configuration of incoming particles, is equally valid. This second interpretation is often preferable in software simulations because it is then not necessary to probe the state of the neighbors of a site in order to determine its next state. All the incoming particle information is stored at the site to be updated. Additionally, only those neighbors affected by the site's next state need to have their state updated to reflect their new incoming particle configuration.

For example, if site i has two particles incoming head-on, we can compute the behavior of these particles after the collision using rule 2Br or 2Bl. After the collision, the state of only two neighbors of site i will be updated to include these two particles as incoming along their links. The other neighbors of site i are not accessed. The new state of site i will be updated similarly by its neighbors. In the case of storing outgoing particle configurations, every neighbor of site i would need to be accessed in order to compute the next state of site i. Thus, the average number of accesses to the array which holds the lattice site information is smaller when the site states are described by incoming particle information.

However, the price of the added performance is that different particle configurations are computed by different update rules because different neighbors are updated depending upon which scattering rule is applied. In the case of a hardware simulation, this leads to greater complexity in the circuitry. Since there is little hardware cost to move all the bits of a state every update because the move

can be done in parallel, we chose to implement our hardware using the outgoing particle definition of "state" and use simpler processors.

It is conceivable that there might be some very clever encoding of the state that would aid in the computation of next state information for the sites. For example, the encoding might provide some information as to the structures that are present at the site: two-body collision, three-body collision, etc. The hope is that the added information in the data encoding would simplify the logic of the update processor, making it smaller and faster. Unfortunately, the size of the data word probably would enlarge, increasing the amount of chip real estate used to hold site values.

We therefore chose an encoding that was natural and compact, assigning a bit to each link from the site. The bit is set to "1" if and only if there is a particle leaving the site along the corresponding link. In addition, there are two more bits, one is set to "1" if and only if there is a particle at rest at the site ("C" for *center*), and the other is set to "1" if and only if the site is to be considered a boundary (solid as opposed to free space, denoted "B" for *boundary*).

The information is encoded in machine byte as follows:

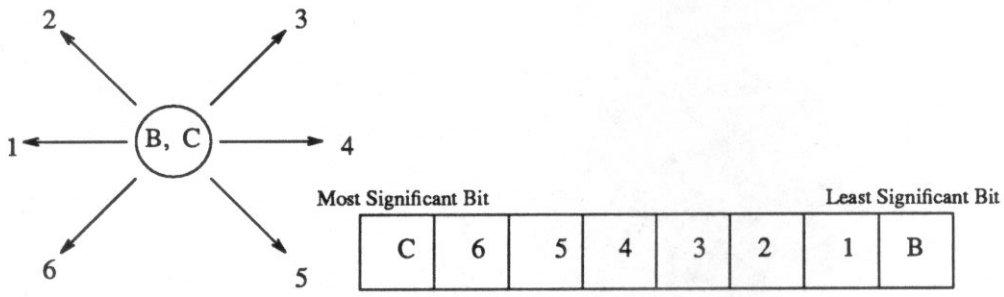


Figure 4-3: Site state encoding

Boundary sites and free-space sites are distinguished by the way that particle interactions take place. In free space, particles collide and scatter according to the rules given in Chapter 1, while any particle that arrives at a boundary site is always reflected back in the direction from which it came. A side effect of this rule is that there cannot be rest particles at a boundary site, for indeed even if there were one, it could not interact with any other particles. We therefore choose not to permit them at all, and the update rules built into the chip enforce this decision.

Sites in a lattice are arranged and numbered according to the following diagram.

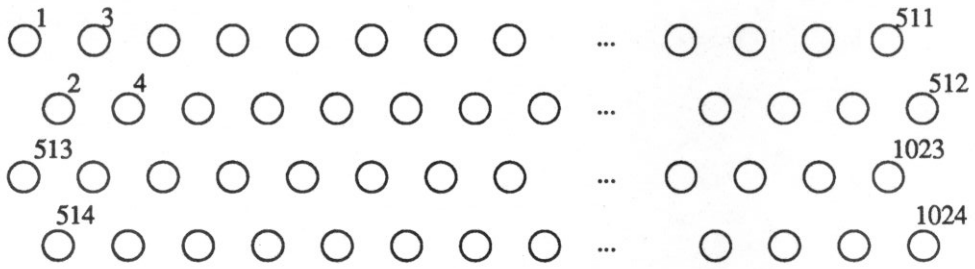


Figure 4-4: Lattice Site Numbering

As the chip data path is 16 bits wide, two sites can enter simultaneously. The chip expects site number one (and all odd numbered sites) to be in the least significant byte and site two (and all even numbered sites) to be in the most significant byte. It generates its output in the same format.

Data in this format can be interpreted in two ways. Referring back to Figure 4-4, the sites numbered 1 through 512 can be considered one row or two. If the former, then the row length is 512, and the simulation can represent any 512 by N lattice. The chip then accepts adjacent lattice sites simultaneously to produce the same. If the latter, two rows, then the row length is 256, and the simulation is on a lattice of size 256 by N. The chip then accepts similarly numbered columns from two separate rows simultaneously to produce data in the same format. Both of these views are consistent, provided the interpretation is maintained during the initialization and post-processing phases of the simulation as well.

4.3. Raster Scan

The data must be serialized in order to be processed by the LGCA chip. The scan is left-to-right, top-to-bottom, and provides the state of two lattices sites each clock period. Refer to Figure 4-5.

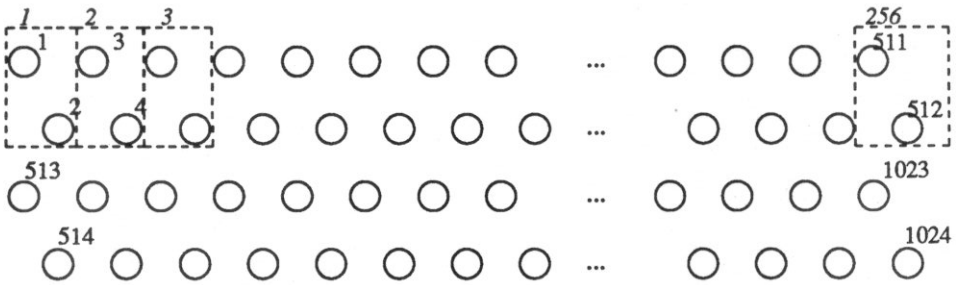


Figure 4-5: Raster Scan Pattern

The data are paired as indicated by the dashed rectangles, with the odd-numbered site value placed in the least significant byte of a 16 bit word. The 16 bit words are sent to the processor array in the order given by the italic numbering shown in the figure.

This scan has advantages over other possibilities. First, it eliminates the need to keep track of the row number. Every other row is identical in its connectivity to adjacent rows. If the data had been grouped (1,3), (5,7), etc., the neighborhoods would differ on a row-to-row basis. Then the neighborhood generator would need to keep track of the row number parity, which would have added unnecessary complexity to its design.

4.4. Architecture and Organization Overview

The underlying architecture of the chip is determined by the wide serial architecture (**WSA**) as described in Chapter 2. In this implementation, the data path is two sites wide. The chip simultaneously accepts the values of two lattice sites and generates new values for two more lattice sites during the same cycle. It can process a lattice either 512 or 256 sites on a side.

A particle collision rule set is chosen from a set of four possibilities by applying control voltages to two pins of the chip, C1 and C0. This permits a small degree of programmability of the device and some control over the viscosity of the lattice gas. C1, C0 interpreted as a binary number determines which set is active. The four particle collision rule sets are numbered (refer to Figure 1-3 for descriptions of the rules 2B, 3S, etc.):

0. 2-body, 3-body, centers (2B,3S,C1,C2)
1. 2-body, 3-body, centers, 3-body-asymmetric (2B,3S,3A,C1,C2)
2. 2-body, 3-body, centers, 4-body (2B,3S,C1,C2,4B)
3. 2-body, 3-body, centers, 3-body-asymmetric, 4-body (2B,3S,C1,C2,3A,4B)

Each chip consists of three principal parts: the shift register memory, the neighborhood generator, and the update processors. The basic architectural plan is shown in Figure 4-6.

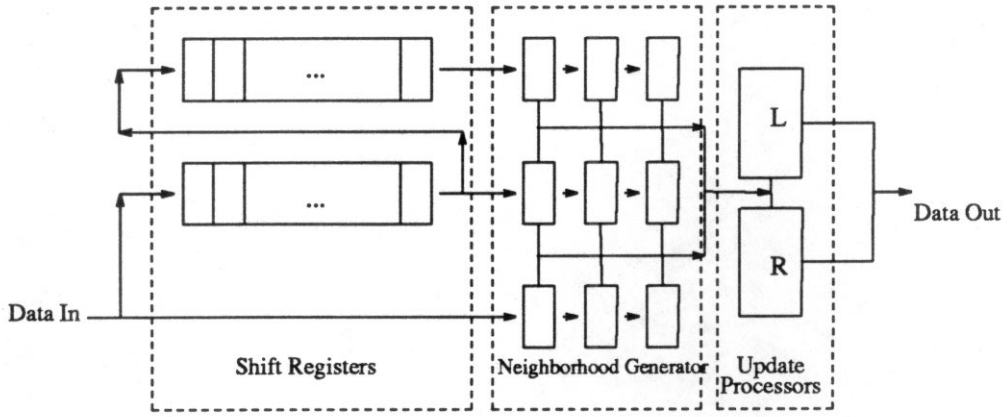


Figure 4-6: Basic Architectural Plan

The shift registers provide delay to permit two-dimensional processing of the one-dimensional data stream. When a data value is entered into a shift register, the registers emit the value of the site in the same column of the previous line. The shift registers are not of programmable length; chip area considerations (see Chapter 2) led to a register length of 256. Each stage in the shift register is wide enough to hold two sites (16 bits) so that the values of 512 sites fit within each of the two shift registers.

The *neighborhood generator* converts *outgoing* particle information from the neighbors of the two sites to be updated into *incoming* particle information for those sites. The neighborhood generator is composed of a set of shift registers in which the required bits are extracted and brought out to the update processors. These processors map from a site's incoming particle configuration to its new state (outgoing particle configuration). This process will be described in more detail later.

The chip contains two update processors, each of which computes a function slightly different than that of the other. This is required because some of the rules (2B, 4B) are probabilistic and have two equally likely outcomes, designated *L* and *R*. Failure to provide both L and R processing will introduce a systematic bias in the simulation and will remove the isotropy that allows the simulation to converge to the Navier-Stokes equation. Whether we choose to select randomly an L or an R outcome when applying rule 2B or 4B, or we choose to distribute the availability of L and R collisions equally about the lattice (spatially) because we expect that these events occur randomly throughout the simulation, no qualitative difference in the simulation will be detectable. Rather than make the processor sophisticated enough to "flip a coin" and behave randomly, we designed each processor to compute

always one of the two possibilities, L and R. Sites numbered (see Figure 4-4) with an even number, will always scatter particles using rules 2Br and 4Br, while sites numbered with an odd number will use the L variants of the basic rules.

With each major clock cycle of the chip, the following activities take place: A new 16 bit word, comprising the states of two lattice sites, is latched into the chip's shift registers and Neighborhood Generator. All other data values in the registers are shifted forward one stage. When this shift is completed, the Neighborhood Generator is then presenting extracted incoming state information for the "current sites" to the Update Processors which place the new site values on the output pins. Because of the latency in the shift registers, the site values emitted are one row above and one column behind those just accepted.

4.5. Hardware Implementation

These architectural ideas are implemented in a straightforward way. Each logical component in Figure 4-6 has a hardware counterpart, but there is additional circuitry to aid testing and debugging. The floor plan of the chip is shown in Figure 4-7.

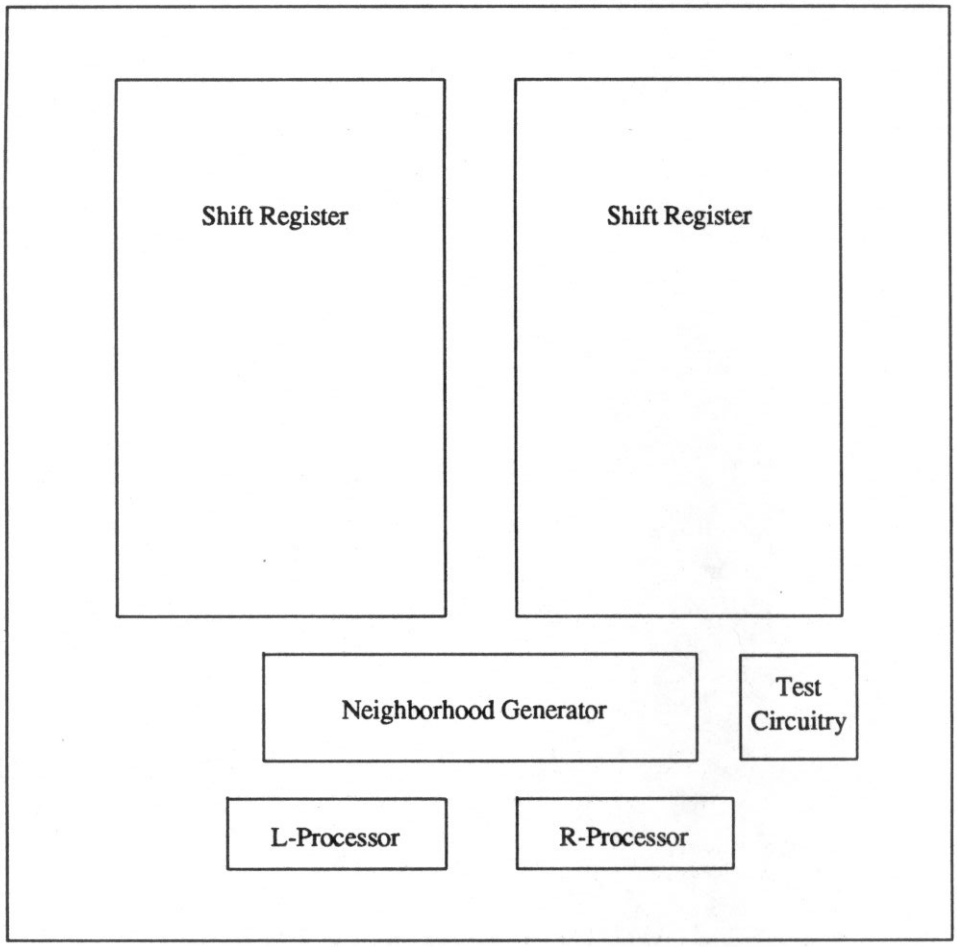


Figure 4-7: Chip floor plan

The majority of the chip real-estate is dedicated to the shift register memory. The neighborhood generator and the update processors are much smaller. The test circuitry is to aid in the testing and verification of the chip behavior.

4.6. Shift Registers

The shift register memory provides delay for the data values that enter the processor. It has 256 stages of 16 bit data words. It is dynamic memory because a static design would have required too much area. The basic cell is shown among the plots in Appendix 1. It is a "master-slave" design, with the master register loaded during ϕ_1 and the slave loaded during ϕ_2. The layout was done by hand, using *magic*, a graphical layout tool, and verified with *esim*, a switch level simulator.

The basic cell was replicated to create a 16 bit wide shift register stage. Copies of this stage were adjoined to create two 128-stage-deep by 16-bit-wide shift registers. These two registers were

placed side-by-side and connected to form the final 256 by 16 bit register. The folding was necessary for one important reason: a linear 256 stage register would not fit on the chip die. As it turned out, folding the shift registers had an added benefit. Since the output of one shift register is connected to the input of the other, the folding placed these two ports in close proximity and shortened the connecting wires. Two of these registers form the upper part of the floor plan shown earlier. The data flow for one of the two shift registers is shown schematically in Figure 4-8.

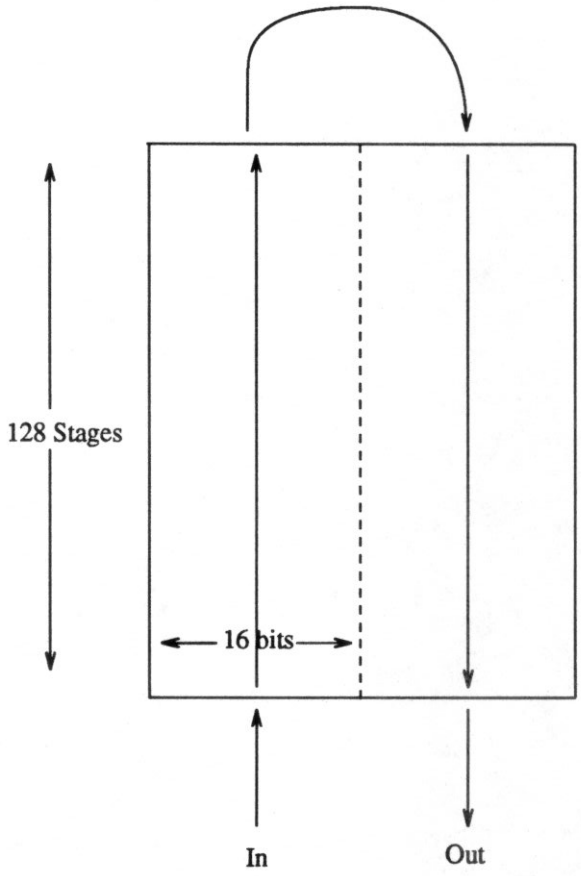


Figure 4-8: Shift Register Data Flow

4.7. Neighborhood Generator

The function of the neighborhood generator is to collect a 3 by 3 neighborhood and convert the outgoing particle information of a site's neighbors into incoming particle information, which is then used to compute the next state. It is composed of a set of shift registers in which the required bits are extracted. To see which bits are required, consider the following diagram (Figure 4-9) in which new states for sites numbered 517 and 518 are to be computed and sites numbered 1031 and 1032 have just been latched as input by the chip.

Let $j:k$ denote bit k of site j. Referring to Figures 4-9, 4-5 and 4-3, we find that site number 517 sees a particle moving toward it on link 1 if 515:4 is 1.

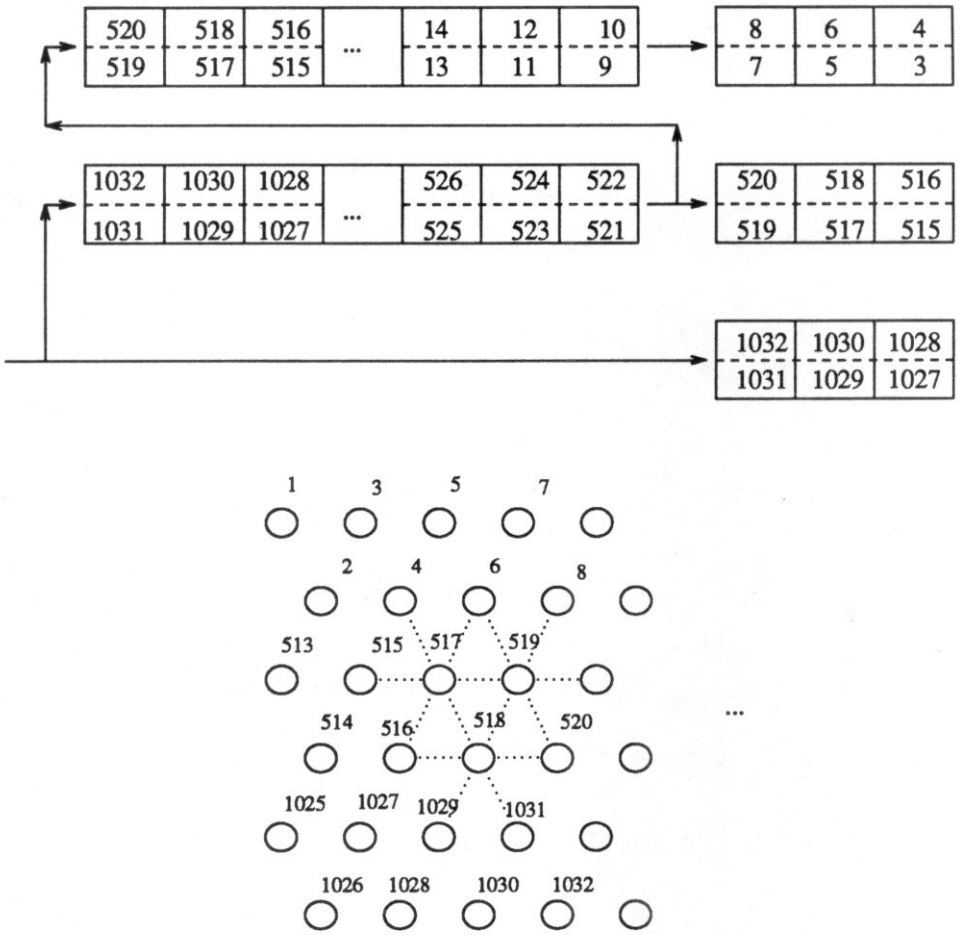


Figure 4-9: Neighborhood Generator Data Dependencies

Similarly for all the other links of site 517. We find that site number 517 depends on 4:5, 6:6, 515:4, 517:0, 517:7, 519:1, 516:3, and 518:2. Site 518 depends on 517:5, 519:6, 516:4, 518:0, 518:7, 520:1, 1029:3, and 1031:2.

Notice that sites 3, 5, 7, 1028, 1030, and 1032 are not involved in these dependencies. They are therefore not stored in the neighborhood generator. Even though sites 8 and 1027 are not used either, site 8 will be required for the next computation, so storage is provided to delay that site value until then. We left storage for site 1027 although it is not strictly necessary.

The bits extracted are sorted and presented to the update processors from which the new state values of sites 517 and 518 are computed. On the next clock cycle all the sites move one stage to the

right, and the process repeats itself. The layout of this section can be found in Appendix 1.

4.8. Update Processors

The update processors map from the input configuration of a site to its output configuration (new state). We wanted to provide some degree of flexibility or programmability over the chip, but at the same time, we wanted to retain the simplicity, ease of design, and small size of a fixed rule implementation.

There is a spectrum of possible structures, ranging from a Turing Machine equivalent processor to combinational logic. The former offers very general programmability and flexibility, whereas the latter is very restricted in the functions that it can compute. What we really want is something between these two extremes.

We chose 4 sets of collision rules which we knew worked by having simulated them. We built a PLA that implemented all of them, and found that this PLA was only twice the size of a PLA that implemented any one of the rule sets. The first PLA is "programmed" by selecting a rule set at the time the inputs are presented. This is accomplished via two control lines, C1 and C0, that determine which rule set number is active at that time.

One PLA performs R rotations for the 2 body and 4 body cases (rules 2Br and 4Br), and the other performs L rotations (rules 2Bl and 4Bl). Their layouts were built by programs, no hand layout was necessary in this case. Two programs, *allL.c* and *allR.c*, were used to generate truth tables for the functions. These truth tables were "minimized" by the program *espresso*, and the results fed to a PLA generator, *mpla*. They use a static structure, with pseudo-nMOS pull-ups for the AND and OR planes.

4.9. Testing Circuitry

The LGCA chip has more than 65,000 transistors. As the vast majority of them are used in the shift register memory, failures and fabrication defects are more likely to show up here than anywhere else. The integrity of the shift register memory is tested by a parity generator which computes the parity of the 16 bit data after it has passed through both shift registers. It is an easy task to determine if there is a "stuck at" fault by sending data through the shift registers and comparing the output of the parity generator with the parity of the input word.

The parity generator is built from a tree of XOR gates. The XOR gate was based on a cell given in [49] but modified slightly. The tree was placed by hand, using *magic* and the functional behavior was verified using *esim*. The XOR cell and the tree are shown in the layouts of Appendix 1.

4.10. Miscellany

Appendix 1 contains the pin-out table and all the layouts for the chip.

Refer to Figure 4-10 for the timing diagram of the chip's major cycle.

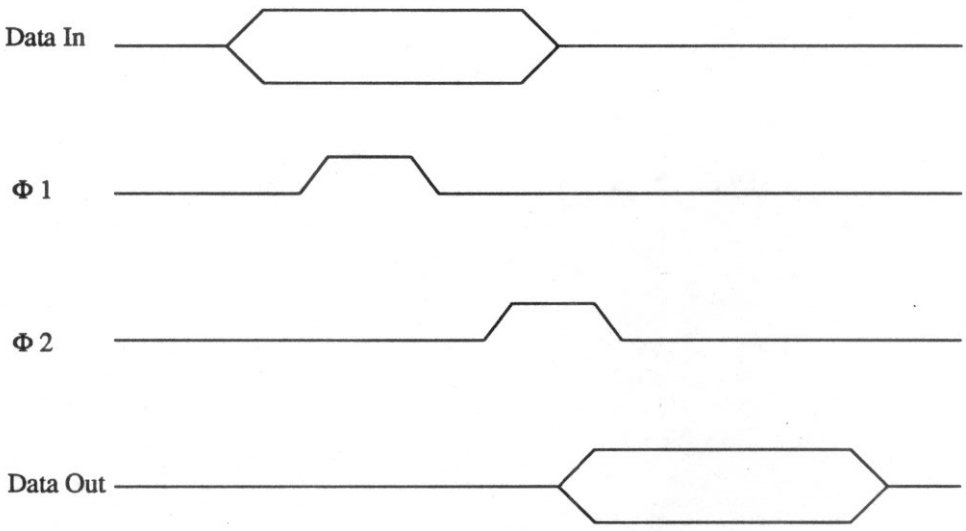


Figure 4-10: Chip Timing Diagram

Data is presented to the chip and must be held until after the falling edge of ϕ_1. The next state values are not available until after the rising edge of ϕ_2 or, if C0 or C1 have been active, not until after they have been stable. ϕ_1 and ϕ_2 must be non-overlapping.

4.11. Testing and Performance

The LGCA chip has been tested on the laboratory bench and as an ancillary processor for a Sun 3 workstation. The yield on the fabrication run from MOSIS was approximately 50%, and all the detected failures were "stuck-ats" in the shift register memory. In all cases, the offending part was detected by attaching an oscilloscope probe to the output of the parity generator circuitry.

The parts were tested to speeds of 50ns per clock phase using a static input data set. However, when a changing input was presented, the parts were reliable only when the phases were at least 70ns, yielding a maximum computation rate of 14 million site updates per second per chip. The power consumption of a chip is less than 100 mW at a clock rate of 5MHz (each phase 100ns).

4.12. Comparison to Other Machines

CAM-6 [25] uses the same basic technique of scanning the two-dimensional array to create a serial stream of data, but it is not a pipelined architecture. It is also designed and tuned for particular host architecture, the IBM PC. Even though it is extensible to larger problems, it cannot provide increased throughput on the same problem. This is where the LGCA chip excels, because it can be pipelined to achieve nearly an arbitrary speedup on the same problem instance.

Sternberg's machines [11, 37] are significantly more complicated, but they are pipelined. Even though it is more powerful than the LGCA, it obtains this power by using more input/output bandwidth. See Chapter 2 and the latter half of [28] for a treatment of computation rate versus input/output bandwidth tradeoffs.

In spirit, our work is closest to that of Broderson and Ruetz [33]. They built separate chips for the various functions of their image processor, including a line delay chip that we implemented as shift register. They concatenated their chips in a pipeline to achieve a final function, which is the composition of the functions of each chip in the pipeline. Since we perform the same function at each stage of the computation, we achieve high performance from chip concatenation, whereas Broderson and Ruetz were forced to make all their chips operate at video rates in order for their system to function at all.

Most of the other work on lattice gas cellular automata implementation has been done in software on general purpose computers. The only directly comparable figures therefore are total throughput. We defer this comparison until the next chapter where we discuss the performance of our prototype system, LGM-1, built around the LGCA chip.

CHAPTER 5
LGM-1, A Prototype Lattice Gas Machine

5.1. Overview

This chapter describes the operation of a prototype system, LGM-1, which hosts the special purpose chips of the previous chapter. The prototype consists of hardware and software that allows a Sun 3/160C workstation to host an LGCA pipeline and use it for the computation of a lattice gas simulation. The prototype is built and tested, and its current configuration is a pipeline of 10 LGCA chips on two boards.

The Sun 3 workstation host does not require a kernel device driver because the user can map the appropriate parts of the prototype into the application program's address space. This allows manipulations which bypass the regular input/output system of the UNIX system and the concomitant overhead. Except for DMA, this is the fastest way we know to perform input/output between the host system and a peripheral. A second benefit is that the prototype can be moved from workstation to workstation without building a new kernel for the host machine.

The prototype system is extensible. Additional cards and chips can be added with a minimum of hardware changes. The original prototype contained only two special purpose chips, but we were able to extend the system to ten LGCA chips with only a small amount of hardware work.

The rest of the chapter will be organized as follows: First, we will describe the hardware of the prototype and its operation. A certain amount of familiarity with the VMEbus, digital logic and hardware construction is assumed. The reader should consult [50-52] for details and descriptions of the omitted background material. Second, we will give the programmer's view of the special purpose hardware and outline the algorithms that produced the flow simulations pictured in a later section of this dissertation.

5.2. Hardware

The Sun 3 uses a triple high VMEbus backplane. We used a VME Specialists master/slave repeater system to extend the Sun's internal bus to an external card cage where we could attach testing equipment to our hardware. We constructed our interface and LGCA chip support hardware within the extended card cage. The basic plan of this system is shown in Figure 5-1.

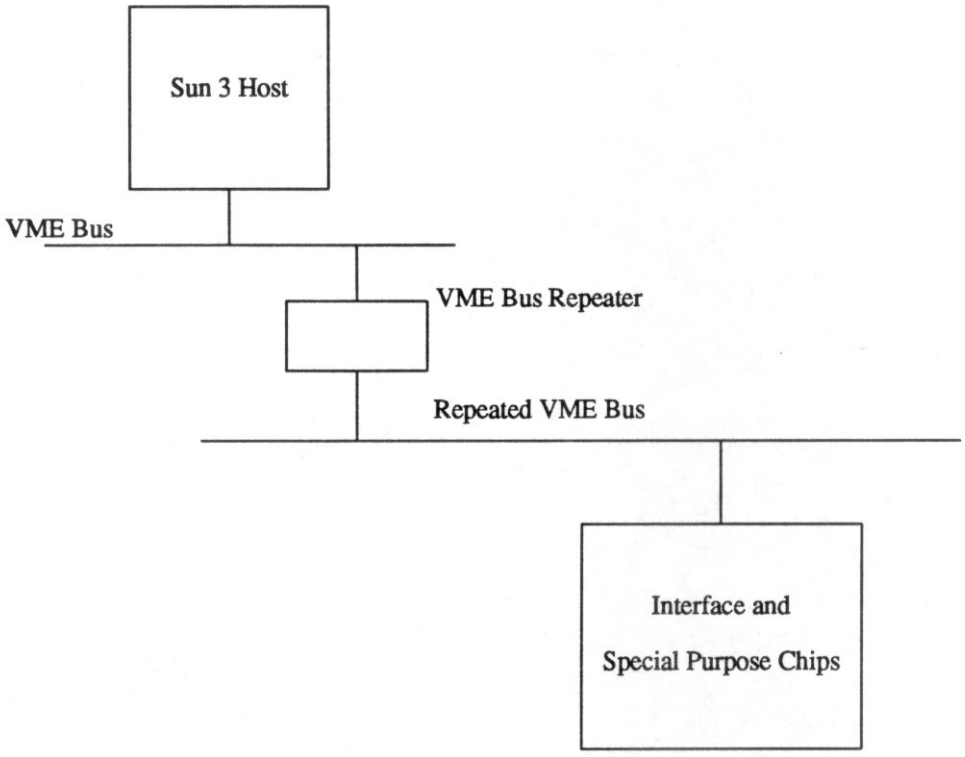


Figure 5-1: Basic Prototype Architectural Plan

The strategy was to have 3 read/write registers: *data_in*, *data_out*, and *control/status*. The application program puts data to be processed into the *data_in* register. The pipeline is started by writing to the *control/status* register. This activates a simple sequencer which generates the required clock phases for the special purpose chips, and latches the output of the pipeline into the output register. The application program can then read the result of the computation from the *data_out* register.

The prototype is a multi-board system. One board houses the VMEbus interface and all the data path and control path logic. It also has room for two of the LGCA chips. Additional boards house buffers and up to eight processor chips each, but they do not interact with the VMEbus directly. They receive their timing signals from the first board.

5.3. Implementation Overview

The VMEbus is an asynchronous bus based on a master/slave paradigm. The bus master generates requests of bus slaves which perform the requested actions and send an acknowledgement of their completion. The VMEbus interface logic performs the signaling necessary to act as a VMEbus slave interface. See the VMEbus interface specification [50] for details on these and other protocol issues.

A set of dual in-line packaged (DIP) switches select the base address to which the interface responds. It is currently set to an unused part of the Sun3 vme24d16 address space (24 bit address, 16 bit data) at physical address 0xdfe000. Appendix 1 has all the schematics of the interface.

5.4. Data Path

The 16 bit *data_in* and *data_out* registers are implemented with edge triggered flip flops. *Data_in* is located at the base address of the board, and *data_out* is located at base+2. The *data_in* register can be switched into the input of the processor array, and the output of the array is fed into the input of *data_out*.

The data path is extendable to other boards via a ribbon cable. The output of the last processor is fed to drivers which lead to a 24 pin ribbon cable socket. A second socket leads to input buffers which then feed the *data_out* register. Single board use is achieved by connecting these two sockets together with a ribbon cable jumper. The data path can be extended to other boards by connecting the output socket of the first to the input socket of the second, and the output socket of the second to the input socket of the first. We used this technique with a second board, consisting solely of LGCA chips and buffers, to build the 10 chip LGM-1 prototype. It is possible to continue the extension of the machine with more boards using this procedure.

5.5. Control Path

The control path is responsible for generating timing signals for the pipeline of LGCA chips and for latching the results of the computation into the *data_out* register. It also provides the control signals C0 and C1 to the array. (These control which rule set the chips will use.) The control path is implemented with flip flops and random logic in TTL.

The control circuitry is synchronous, based on a clock generated by dividing the 16.7 MHz bus clock which is provided by the bus master (Sun3). A finite state machine generates timing information in response to a start signal which is provided by the controlling application program.

The control and status register is located at the base address plus 4. Bit 0 of this 16 bit register, the least significant bit, is the start signal. When this is set to 1 by the application program, the state machine is started and puts the array through one complete major cycle, and then latches the result into

the *data_out* register. The next two significant bits of the control register, bits 1 and 2, hold C0 and C1 respectively. These two control signals are delivered to all of the processor chips in the array.

The state machine automatically clears the start signal bit in the control register so that when the cycle is complete, the machine is not retriggered. It also guarantees that the two clock phases, ϕ_1 and ϕ_2 are non-overlapping. The data paths of the *data_in* and *data_out* registers are switched so that the path to the processor array is active instead of the path to the VMEbus. This eliminates possible contention between the two systems.

ϕ_1, ϕ_2, C1, and C0 are buffered and brought to the ribbon cable connector. This provides timing and control signals for additional boards of processors. The additional boards buffer the incoming signals, and the clocks are further buffered every four additional processor chips. The data bits do not need additional buffering because each processor chip only drives one load: its neighbor.

See Appendix 1 for all schematics, timing diagrams, and state machine specifications.

5.6. Programming Model

The programmer sees only the three registers described above. They are memory-mapped onto a structure in the user process address space. This structure is defined by the C declaration: `#include <sfmreg.h>` which contains the following:

```
#define START    0x0001

/* a structure that matches the actual board */

struct sfmdevmem {
    unsigned short data_in;      /* data in register */
    unsigned short data_out;     /* data out register */
    unsigned short s_ctrl;       /* status and control register */
    char wasted_space[8186];     /* fill out the 8k page */
};
```

The basic algorithm used to operate the processor array is:

data_in = 2site_values;
status_control = START;
2new_site_values = data_out;

A large simulation uses this as an inner loop in each of 3 stages. The first stage fills up the processor pipeline latency. Array data is sent to the pipeline, but no new data is produced because the pipeline is empty. During the second stage, the pipeline is full, so every data word sent to the pipeline allows the computation of two new site values. The third stage begins once all of the old lattice data has been entered. There are still data points in the pipeline, so we force them through by sending

random sites with particle distributions given by Equation 1-1. The latency of the pipeline is a function of the number of chips in it. The program therefore has the following structure:

```
BEGIN
initialize array of sites according to uniform velocity distribution

set boundary bits in selected array cells

while (current generation < last generation) do
    pre-load array with uniform velocity field
    compute latency of processor array (function of number of chips)
    while (latency not filled) do /* stage 1 -- fill latency */
        data_in = array_data;
        status_control = START;
        array_data = data_out;
    od
    while (still old data to send) do   /* stage 2 -- send and receive data */
        data_in = array_data;
        status_control = START;
        array_data = data_out;
    od
    while (still not received all new data back) do   /* stage 3 -- flush pipeline */
        data_in = random_site;
        status_control = START;
        array_data = data_out;
    od
od
END
```

The treatment of the sides of the lattice is subtle, but very important. Remember from Chapter 1 that if the finite lattice is to be made to look infinite, then the links to the sites on the sides of the lattice should see particles that are consistent with a fluid flow of the simulated velocity. Failure to observe this will guarantee catastrophic failure of the simulation.

For example, if the sites at the sides of the lattice see no incoming particles, then in essence, the lattice sees a vacuum on all sides. The response of the system will be the natural one: all the particles (fluid) will flow out on all sides to try to fill the vacuum. Naturally, this will disrupt whatever the simulation is trying show.

Another example of how the simulation might fail is if the incoming links to sites on the sides of the lattice are set to particular, unchanging values. These settings may have been derived from Equation 1-1 (the particle distribution equation) by flipping a biased coin, but once the links are set they are never changed or are changed infrequently. This again introduces fixed biases in the distribution of particles "outside" the lattice, and the lattice will not "see" the distribution of particles that would indicate that it is embedded in an infinite array of fluid.

We have empirically determined that if the side link conditions are changed at least every 10 generations of the simulation, then the simulation will behave properly. If the particle occupancy were

changed every generation, the results are slightly different and measurable, but are not significant for our purposes.

During the time that the lattice data is in the processor pipeline, the data are not available for update. Since each chip in the pipeline contributes one further generation of computation, the edge update criterion implies that the maximum depth of the pipeline is ten and the prototype hardware has this limit.

We will discuss a possible hardware solution to this problem in Chapter 7, but software manipulations can ameliorate the ill effects of infrequent boundary influx. The sides of the lattice effectively are wrapped around because of the way the raster scan is done. We want to isolate these edges from one another, preventing any correlation of behavior from one side of the lattice to the other. We do this by changing the values of all sites near the sides of the lattice before each pass through the processor pipeline. The top and bottom sides do not require this treatment because random sites conforming to the probabilities given in Equation 1-1 are introduced to fill the pipeline before real computation takes place, and to flush the pipeline after the last lattice site value is introduced.

5.7. Graphic Display

The graphics display software was written by undergraduate Mark Taylor as part of his senior independent work [53]. It runs under Sunview (Sun 3 windowing system) and performs post-processing of the raw lattice data into hydrodynamic cells and graphic display of the resulting hydrodynamic velocity fields. It supports arbitrary lattice dimensions, hydrodynamic cell size, number of frames, and frame of reference.

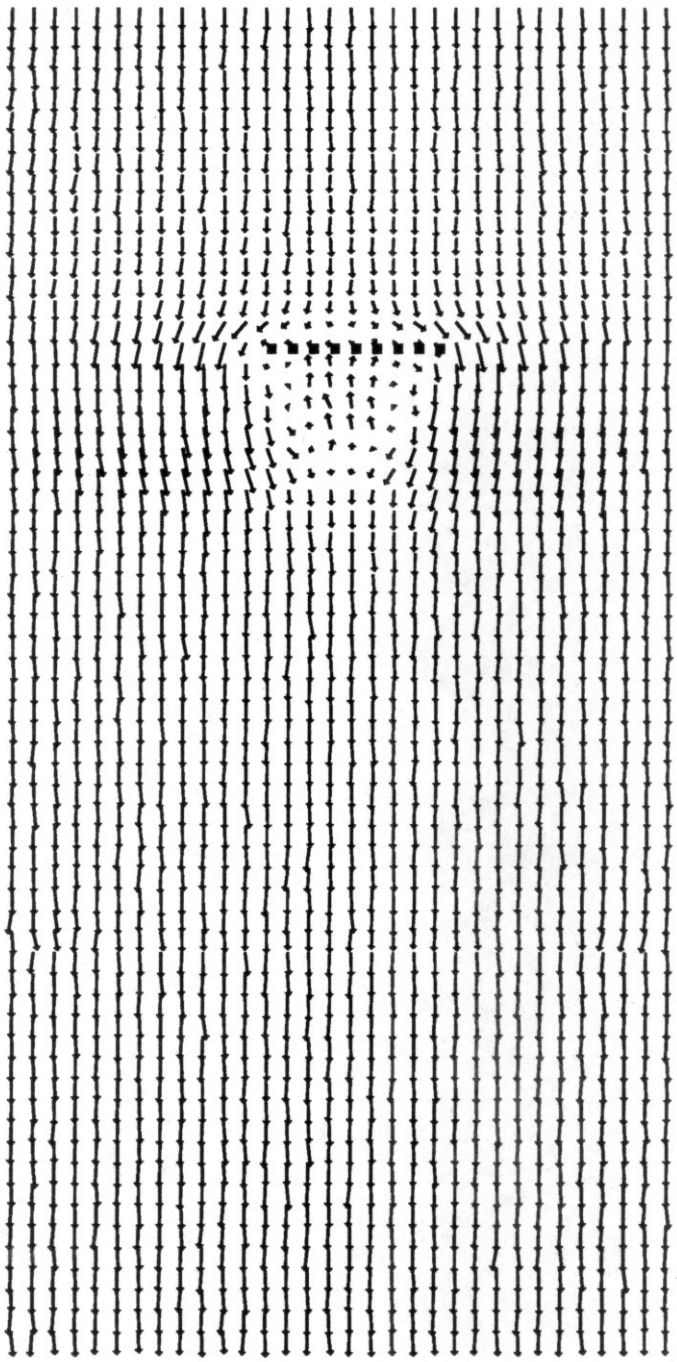
5.8. Performance Levels Of Current System

The application program is heavily I/O bound. It can only start the array once every 3 microseconds, yielding a throughput of 7 million site updates per second with a pipeline that is 10 chips deep. The top hardware speed is 14 million site updates per second per chip, so only a small percentage of the maximum performance is realized.

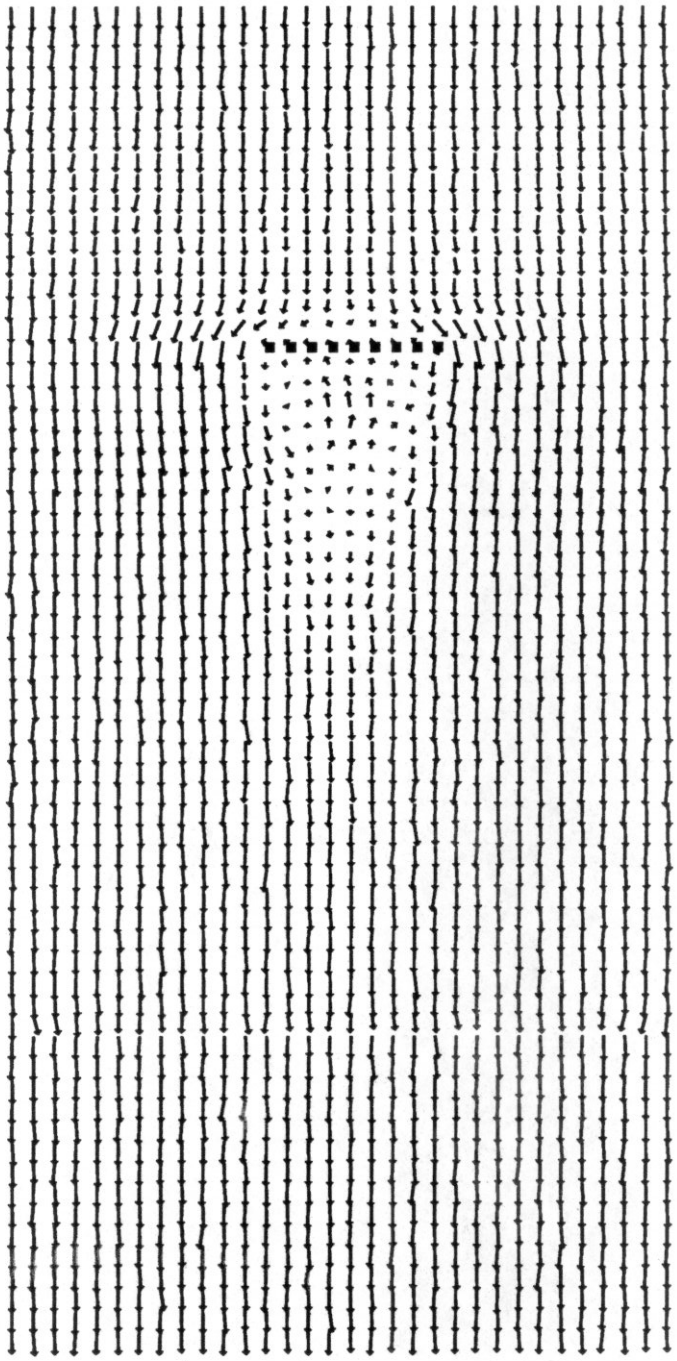
The machine was operated continuously for more than a week without any detected hardware failures. We performed simulations of flows over variously shaped objects, and examples of the results are shown in the following figures. All the simulations shown were performed on the ten chip LGM-1 system, where the total throughput was approximately 7 million site updates per second. The particle density in each case was 0.2, which we know to yield good results and a relatively small fluid viscosity. We estimate the Reynolds number at 40 for these simulations.

The figures show the velocity field of a flow simulation, and the flow is always from top to bottom. A *bias velocity* is a vector subtracted from every hydro cell velocity vector and the effect of this is to shift the frame of reference of the viewer. For example, if the bias velocity is zero, the frame of

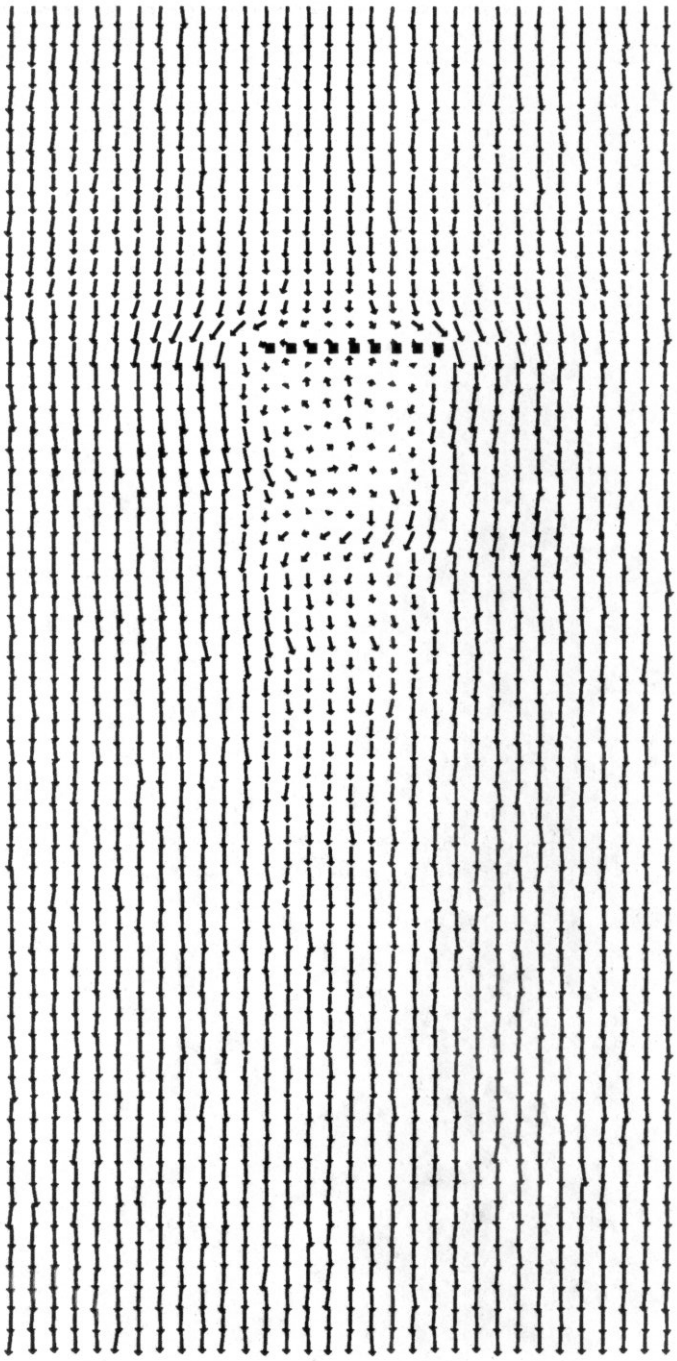
reference is that of the object at rest with the fluid moving past it. If the bias velocity is equal to the speed of the majority of the fluid, then the fluid is at rest and the object is seen to be moving through the fluid. Intermediate values of the bias velocity yield other reference frames where different structures of the flow will be more easily visible.



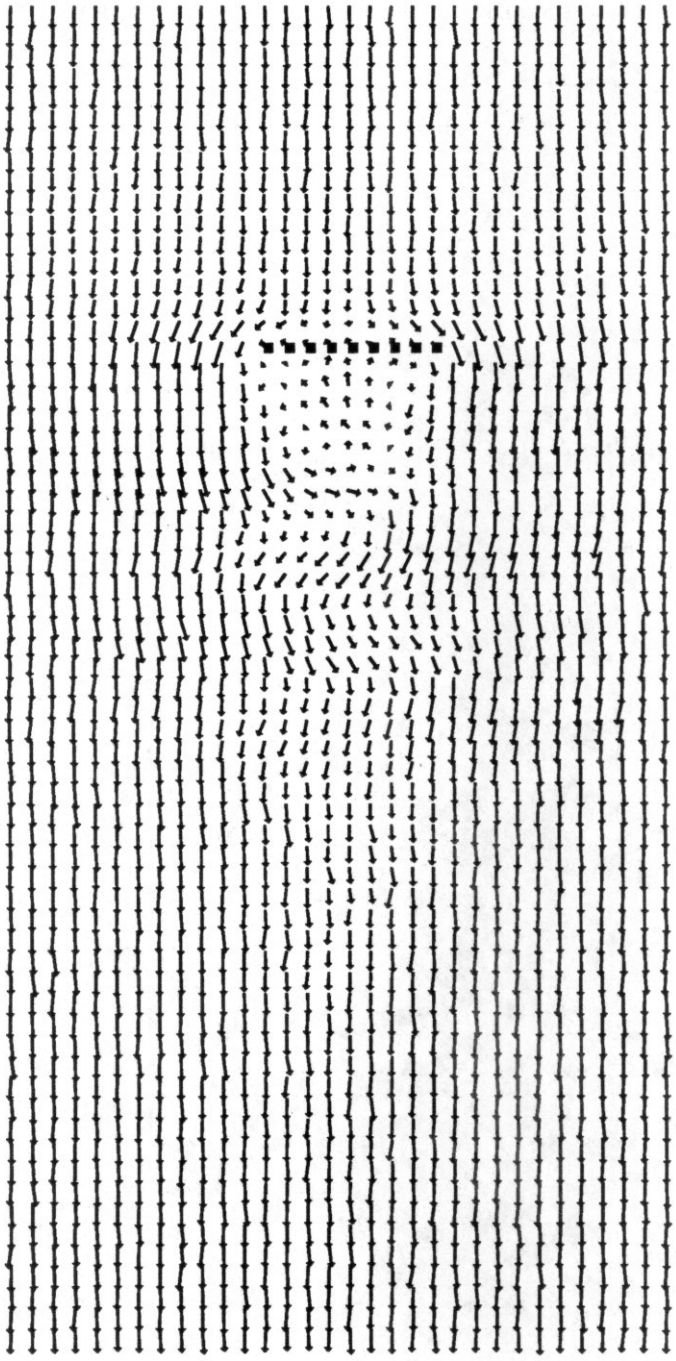
Bluff Plate
512 by 1024 sites
Velocity = 0.55
Bias Velocity = 0.00
Generations = 1000



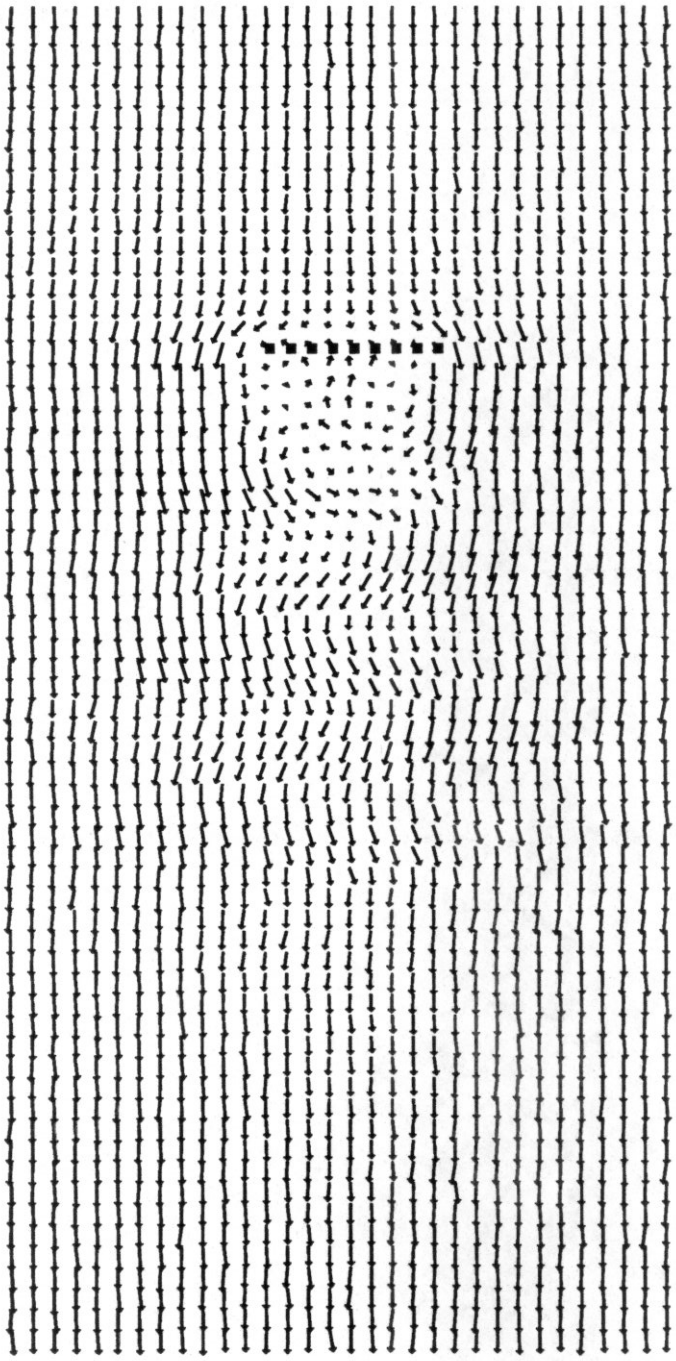
Bluff Plate
512 by 1024 sites
Velocity = 0.55
Bias Velocity = 0.00
Generations = 2000



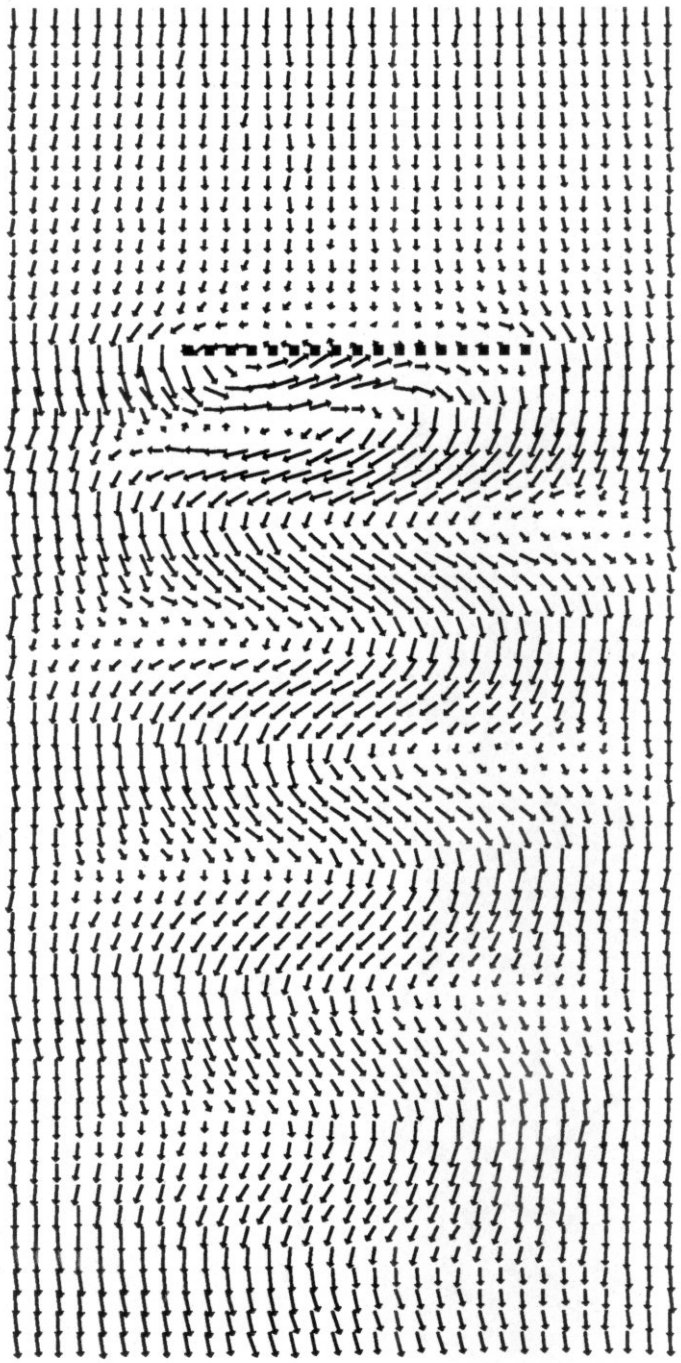
Bluff Plate
512 by 1024 sites
Velocity = 0.55
Bias Velocity = 0.00
Generations = 3000



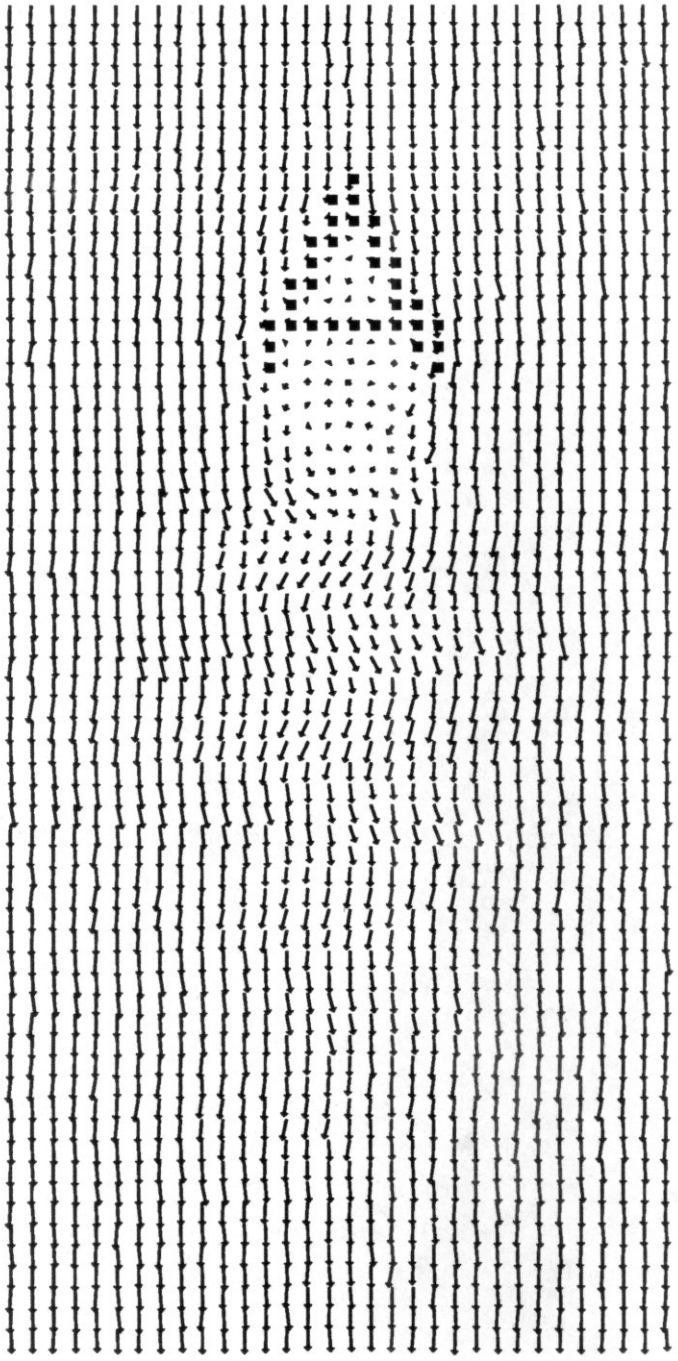
Bluff Plate
512 by 1024 sites
Velocity = 0.55
Bias Velocity = 0.00
Generations = 4000



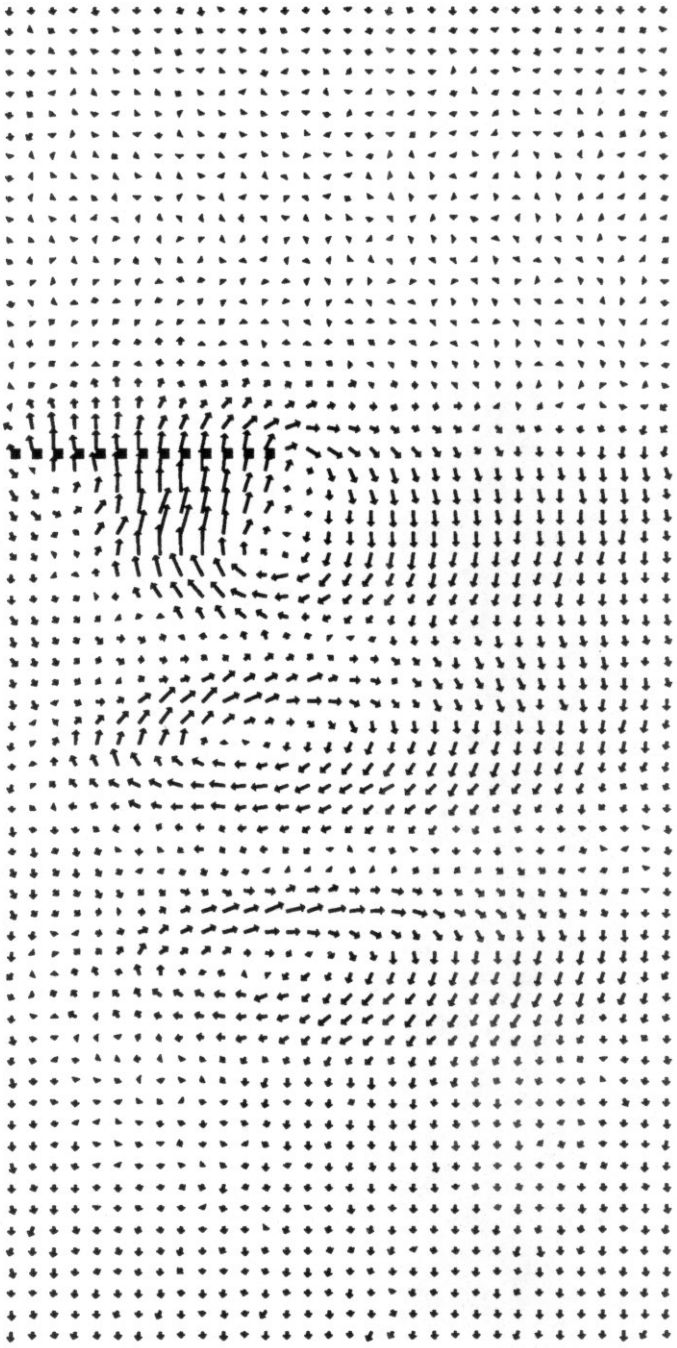
Bluff Plate
512 by 1024 sites
Velocity = 0.55
Bias Velocity = 0.00
Generations = 5000



Bluff Plate
512 by 1024 sites
Velocity = 0.55
Bias Velocity = 0.00
Generations = 10000



Wedge
512 by 1024 sites
Velocity = 0.55
Bias Velocity = 0.00
Generations = 5000



Jut
512 by 1024 sites
Velocity = 0.55
Bias Velocity = 0.40
Generations = 300

Despite its small size and modest support interface, the LGM-1 is one of the fastest machines in the world for the computation of the FHP lattice gas. Hayot, et al. [54], discuss the implementation and performance of the best known algorithms for the FHP lattice-gas software simulation as executed on several different general purpose machines. They also extrapolate to predict the performance of larger sibling computers on the same problem.

Hayot reports a site update rate of 2.3 million sites per second on a 32-processor INTEL iPSC hypercube. An Alliant FX-8 shared memory vector supercomputer gave only 0.7 million sites per second on the same problem because a large part of the code had not been vectorized nor made concurrent. We summarize the reported measured performance of various machines in the following table (from [54]). Even though the lattice sizes differ, the comparison is fair because the performance of these machines is independent of the size of the problem.

System	Lattice Problem Size	Performance, site updates/sec
32-processor iPSC	4096x4096	2.3×10^6
Alliant FX-8	4096x4096	0.7×10^6
CAM-6	256x256	4.0×10^6
GAS (built at ENS Paris)	512x256	6.5×10^6
Connection Machine	4096x4096	10^9
LGM-1	512x1024	7×10^6

Table 5-1: Reported Actual FHP Performance

The GAS machine is a dedicated FHP lattice gas machine designed and built at Ecole Normale Supérieure in Paris. No published material is available for this machine. The Connection Machine is a general purpose computer composed of 65536 one-bit processors arranged in a multi-dimensional hypercube [55].

Hayot, et al., were able to predict the performance of more powerful hypercube processor systems by extrapolation. They report a predicted performance of 86 million sites per second on a 128-processor INTEL iPSC-VX, and 120 million sites per second on a 1024-processor N Cube-10. The LGM-1 has a theoretical maximum performance of 140 million sites per second with 10 chips, but the prototype implementation used only a small fraction of this power. The reduced performance is caused by a having a slow host, and a simple, memory-mapped system interface.

It would be possible to tune the program that controls the chips for higher performance, but this would probably only offer a marginal improvement in throughput. Similarly, the entire prototype system could be moved to a faster Sun 3 workstation. This might offer a doubling of speed, but the real gains would come from being able to deepen the processor pipeline.

As we have already stated, a problem in our implementation starts to occur if the pipeline is deepened beyond 10 chips. If the sides of the lattice are not recomputed, then the particle distribution becomes seriously biased and the simulation does not converge. We must recompute the edge sites inside the pipeline because these intermediate generations are not otherwise accessible. Because the current processors are not capable of doing this, we need a new processor that can compute a random site value that conforms to the probabilities given by Equation 1-1.

The design of this "boundary processor" is described more fully in Chapter 7. We [56] already have sections of it built in TTL and tested. We expect that a working prototype of it will be ready for installation in the current processor pipeline in several months.

CHAPTER 6

A Proposed Environment For Fast Prototyping Of Linear Systolic Processors

6.1. Overview

The LGM-1 prototype we constructed was a success from several standpoints. First, the LGM-1 was instrumental in allowing us to test and verify the special purpose chip. It provided an environment in which the chip could be exercised by providing input patterns and by comparing the results with known correct output. Its biggest shortcoming was the input/output bottleneck that limited system speed when the system was debugged and ready for production use.

In this chapter we propose a host support system that is conducive to fast prototype implementation of pipelined arrays and other special purpose structures with custom outboard devices. The environment includes both the hardware support and a programming model that enables chip designers to debug and operate their creations. The proposed hardware interface is standardized to simplify the construction of the prototyping environment, but would not necessarily force common chip pin-out conventions on the designer. The programming model is an illusion created by a subroutine library that provides entry points for data format conversion, data input, data output, and device control.

The proposal of this prototyping environment is motivated primarily from our observations of VLSI hardware projects undertaken in the Department of Computer Science over the last 3 years and our experience with this project. Only two projects culminated in the operation of a working production prototype of some sort. Many projects never even reached this stage; their chips remain untested.

In 1984, Steiglitz, et al. [35], designed and fabricated a chip that simulated a one-dimensional cellular automaton. The chip was tested in a primitive bread board environment by exploiting a priori knowledge about the "fixed points" in the machine's functions. Because a production level system was never built, the full power of the chip was never exploited to do useful computation.

In 1986, Lopresti built the PNAC (Princeton Nucleic Acid Comparator) nMOS chip [57]. In this case, the chip was eventually incorporated into a production level support board (multibus SUN-2) and

tested. The performance of this system was impressive, but was still far less than the capabilities of the custom chip. Since that time, PNAC has been installed in other computer systems (IBM PC AT in progress). Even though the second generation designs have included "tricks" that allow the custom chips to be operated more efficiently, the PNAC chips on the board are still grossly underutilized by the host system.

To some extent, our project has suffered from the same failings. Our hardware prototype offers about 60 to 100 times more performance than software simulation on a Digital Equipment Corporation Vax 8650, but this is still two orders of magnitude less performance than the custom chips themselves can provide.

Each of the aforementioned projects has several common elements or aspects. They are all pipeline processors or linear systolic arrays, and each operates according to the same inner loop:

step (a). Write data to chip.
step (b). Start chip.
step (c). Read result from chip.

However, each of these projects had differences in data format, word size, number of ports, and chip pin-out conventions. This precluded the use of the same interface board for each. The designer was forced to build interface hardware for a particular target host and support hardware that was tuned for the particular special purpose chip. This entailed re-inventing known technology: interface handshaking circuits, address decoders, etc. In reality, all that was desired was a hardware platform that would make it easy to write software to implement the basic loop just described.

In some circumstances, a VLSI chip can be an end in itself, rather than a means to achieve a larger scale goal. In these cases, the chips often go unused, even untested. Our viewpoint is, however, that a custom chip design needs to be integrated in a larger system, and that the goal is enhanced performance for a special class of problem. Nevertheless, it is true that less is learned by building an interface board than is learned by building some new, previously unstudied architecture. But the fact remains, without the scaffolding, an isolated chip is of limited utility.

A related issue is whether the ideal prototyping environment should free the chip designer from additional hardware work, allowing him or her to plug the custom chip into an existing framework without doing any further wiring. The complexity of such an environment would be formidable. We argue that it is more sensible to build something simple that provides a wealth of generic services which the chip designer can exploit. After all, if he or she is comfortable with hardware, and if the time necessary to install the chip in the interfacing and support environment is modest, then it will not be an impediment to complete system integration. The advocates of a "plug it in and go" support system tend to underestimate the hardware work needed to build the support system and software work

that would be involved in configuring the system to a new custom chip.

The prototyping system must meet the needs of current and future VLSI based linear systolic array and pipeline processor projects without modification. It also must allow the easy addition of custom chips to lengthen the array or deepen the pipeline. We intend to use this system to test our theories about the indefinite extension of processor pipelines. It may be more important to provide the facility to add hardware than it is to make each stage of the system operate extremely fast.

To summarize the requirements, the prototype system should satisfy all the needs of all linear systolic and pipeline projects. It should support different word sizes at each of a number of data input and output ports, as well as flexible timing and clock generation. The clock rate, and therefore the system throughput, should be adjustable via software. The pin-out conventions of the experimental device should not be fixed, but the amount of hardware work necessary for the installation of the special purpose devices should be minimal.

Another consideration is the implementation of the prototype system itself. It must be feasible to build and test with our current hardware configuration and local expertise. This rules out technology such as microprogrammed sequencers for on-board state machine control because we lack both the equipment and the experience to construct systems from these parts. Commercial interface and prototyping boards may be of some help, but we have found them to be of low quality and reliability.

Some thought was given to changing the the interface paradigm from memory mapped to direct memory access (DMA). The hope was that performance could be increased by as much as a factor of 10 by eliminating the CPU from the memory input/output data path. The increased complexity is not justified for two reasons. First, performance can be increased to any arbitrary level by increasing the number of special purpose chips in the pipeline. Second, a DMA system *requires* a device driver, and is therefore not as transportable as a memory mapped system. We therefore retain the memory mapped paradigm, relying on the possibility of adding additional special purpose chips to make up for the performance differential.

Another contentious question is what computer system should act as the host to this prototyping environment. Some have argued that the IBM PC/AT is a natural choice because it is common, and because the programmer has full control over the machine. We disagree. The lack of a real operating system for these machines is a serious liability, and for the purposes of doing research, the services provided by a machine like a Sun workstation and an operating system like Unix are very desirable. The objection usually raised against doing development under Unix is that a device driver is necessary and the virtual memory management system makes using absolute hardware addresses difficult. We have shown in our previous work that neither of these objections is valid, and we therefore adopt the Sun 3 workstation and Unix as the host to our hypothetical prototyping system.

What follows is a more specific proposal for a prototyping environment for linear systolic and pipelined processors. This hardware would be useful for the continuation of this project, and possibly

for future VLSI projects. Since none of this hardware has been constructed, we have left some details in a more or less vague form. It is difficult to anticipate features that may be desirable, but once detailed design and construction has begun, some of the details may be become obvious or constrained.

6.2. Hardware Specification

The prototyping system occupies 8K in the virtual address space of the host, chosen to coincide with the hardware page size of the Sun 3. The hardware address is switch-selectable and may begin on any page boundary within either the 24 bit address space and the 32 bit address space VME bus. There are eight read/write 32 bit data registers which are divided equally between input and output. Four input registers, located at addresses 0, 4, 8 and 16, provide data to the special purpose chips. Similarly, four output registers, located at addresses 20, 24, 28 and 32, receive data from the processor array. The designer is free to use as many or as few of the bits in the data path as desired.

There are three control registers. The first, located at address 36, controls the clock generator period. The value stored in this register is used as part of a programmable divide-by-*n* circuit which creates a time base for clock and timing signals. The second register, located at address 40, is the status and control register for the state machine. Its bits provide the **start** signal to run the special purpose chips and to latch results into the output registers. Other bits indicate whether the machine is idle or running. The third control register is for special purpose chip control. It provides 32 bits for use on the control pins (if any) of the special purpose chips. It is located at address 44.

All the registers are read/write, and must be addressed as words (32 bit quantities) on word boundaries.

The other major hardware unit is the clock generator. It is responsible for the synthesis of clock and timing signals, some of which are provided to the custom chips, and others which are used to control the underlying state machine. It can be programmed to produce a clock rate between 50ns and 500 ns per clock phase in 25 ns steps. All the generated phases will have the same period. Our suggested implementation is to use a programmable divider to divide a free running clock based on a quartz crystal oscillator.

Most of the usual prototype circuit specificity occurs in the clocking and latching circuitry, so naturally this is where we want to provide the greatest possible generality. The clock generator provides 2, 4, 8, or 16 non-overlapping clock phases. The developer can use them to drive clock pins and to latch data from the chips into the output registers. Initially, we suggest that a jumper block select the number of clock phases and which phase is used to latch each output register. It is conceivable that these functions might be moved to programmable switches in the future, and there is ample room remaining in the address space to accommodate the requisite control registers.

6.3. Software Specification

We anticipate minimal information hiding. A C *struct* defines the offsets of the various registers and constants and masks that are useful in starting the machine and determining its status.

```
struct protodevmem {
    unsigned int ir_0;          /* input registers */
    unsigned int ir_1;
    unsigned int ir_2;
    unsigned int ir_3;

    unsigned int or_0;          /* output registers */
    unsigned int or_1;
    unsigned int or_2;
    unsigned int or_3;

    unsigned int ck_per;        /* clock period control */
    unsigned int ctl_state;     /* state machine control and status */
    unsigned int ctl_user;      /* user control word */

    char filler[8148];          /* use up the 8K page */
};
```

In addition, there are a few skeleton macros and functions that simplify the programmer's job. They and their functions are given in Table 6-1:

Functions and Macros

Name	Action
wdata(port, value)	Writes data specified by value to designated port. Value is treated as a 32 bit unsigned quantity.
rdata(port,)	Reads data from designated port. Returns an unsigned int.
clkper(nanos)	Set the clock period to the requested number of nanoseconds. The requested period will be rounded to the nearest 10 nanoseconds.
clk()	Run the processor array for one multi-phase major cycle.
clkstat()	Returns **1** if the processor array is running, **0** if idle.

Table 6-1: Software Entry Points

6.4. Physical Specification

The prototyping system should be constructed on double high VME cards. We highly recommend that the internal Sun 3 VME bus be repeated by a suitable commercial repeater system to an external card cage. This will provide a degree of safety for the developer and the Sun 3 host by

providing physical access for the former and electrical isolation for the latter.

The proposed prototyping environment is a multiple board system comprising two basic board types: type M, and type S. There is only one type M board, the master control board, in a system, but there can be an arbitrary number of type S (slave) boards. The type M board would contain the VME interface logic with full 32 bit address recognition, and all of the 32 bit registers and the state machine circuitry. There would also be a set of differential line drivers to drive twisted pair ribbon cable that carries data and control signals to the type S cards. The type M card also would contain a set of differential receivers to allow type S cards to return results to be latched in the output registers. The drivers on the master board should transmit the contents of the 4 input registers, the 16 bits of clock phase, and the 32 bits of device control. The receivers would buffer four 32 bit quantities which are to be latched into the output registers, and 16 bits of clock.

Separate data and clock cabling should be used so that different distributions topologies can be used for each type of signal. For example, the developer might want to use a tree for the clocks to keep skew small, but use a linear bus for the data lines. Twisted pair cable is used to provide immunity from noise. The master should also be able to receive clock signals so that latching signals are not skewed with respect to the incoming data signals (which come from another board).

The second type of board, type S, is a slave of the type M board. It would contain differential receivers and transmitters and the special purpose chips. The receivers would buffer four 32 bit data words, one 32 bit control word, and the 16 bits of clock. The incoming data words should be provided to a special purpose chip, and the clocks distributed as desired. Differential transmitters would be provided to chain type S boards together. The clocks and control bits would be propagated automatically from the receivers to the transmitters, but the data bits would come from the output of one (or more) of the special purpose chips. The clock signals would be propagated to two output connectors to allow clocks to be distributed as binary trees, if desired.

6.5. Final Comments

The developer is presented with this raw environment. It provides some organization and structure, but hardware work remains before the designer's chips can become operational. We have one implementation suggestions that could accelerate the process. The designer should develop another board which has all the chip-specific wiring but has the connectors that mate with the type S board connectors. The chip-specific board has no logic; it only contains the wire routing to provide data and clocks to the special purpose chips. Once this is developed and produced, it is a simple matter to plug all the pieces together to arrive at a working system.

The separation of functionality has an added benefit beyond providing a simple hardware interface. It is also bus and host independent and this makes it simple to migrate the special purpose chips to a new host. The design of a type M board is all that is required. We grant that this might not be an

inconsequential amount of work, but it is restricted to a portion of the system and this in itself is simpler than the traditional, chip-specific prototyping technique.

Chapter 7
Conclusions and Future Work

7.1. Summary and Conclusions

This dissertation has explored theoretical and practical questions in the design of massively parallel machines for regular, iterative, spatially distributed problems. We have analyzed and compared two architectures that are efficient for lattice computations and that are suitable for VLSI implementation. We also have studied the effect of clock skew as a possible limitation on the ultimate performance of large, globally synchronized multi-processing systems. This work culminated in the design of a special purpose VLSI CMOS chip and the construction of a working prototype machine for the Frisch Hasslacher and Pomeau (FHP) lattice-gas model [17].

Part I (Chapters 2 and 3) of this dissertation has addressed two different theoretical issues. In Chapter 2, we have given an analysis method that evaluates VLSI architectures geared toward lattice computation based on size-throughput and area-bandwidth (number of pins) tradeoffs. Using this technique, we analyzed and compared two competing architectures (**SPA** and **WSA**) for the two dimensional lattice computation problem. This analysis convinced us to build a chip based on the **WSA** architecture.

The **WSA** architecture has a property that we call *linear speedup*. If n processors, which are fixed in size and cost, provide n times the throughput of one processor on the same problem instance, we say that the processor provides *linear speedup*. Other lattice processing architectures, such as the CAM-6 [25] and the PE-1 [4] lack this important property. Speedup is achieved with these architectures *only* when the problem instance size is also increased. We consider this to be a disadvantage of these architectures.

A consequence of this simple idea is that under certain circumstances it may be more profitable to make the machine larger than it is to increase its clock rate. The reason why is simple: a performance differential between a system with a fast clock and a system with a slow clock can be equalized by adding more hardware to the slower system at linear cost, but with linear speedup. This fact motivated our discussion in Chapter 6 of a prototyping environment in which it is easy to add

hardware.

Given these properties, it was natural to ask if there would be a theoretical limit to the length to which a linear array can be extended. Our hypothesis is that the length would be restricted because of either timing problems or reliability and we suspect that power density or power dissipation would not be a limiting factor. We began to answer this question by investigating the problem of timing. We presented this work in Chapter 3 where we proposed and analyzed a probabilistic model for clock skew accumulation based on variations in buffer and wire delays.

We applied these results to two different models for clock distribution. The first, called *metric-free*, dictates that the skew in a buffer stage is Gaussian with a variance independent of wire length. In this case the upper bound on expected skew grows as $\Theta(\log N)$ for a system with N processing elements. In the the second model, called *metric*, the clock skew in a stage is Gaussian with a variance proportional to wire length, and the distribution tree is an H-tree embedded in the plane. In this case the upper bound on expected skew is $\Theta(\sqrt{N \log N})$ for a system with N processors. Thus the probabilistic model is more optimistic than the deterministic summation model of Fisher and Kung [27], which predicts a clock skew of $\Theta(N)$ in this case, and is also consistent with their lower bound of $\Omega(\sqrt{N})$ for planar embeddings.

Further work in this area will include the analysis of other clock distribution topologies and the inclusion of the effects of processor interconnection graphs. In addition, study must inevitably turn to the effects of reliability.

Part II of this dissertation (Chapters 4 through 6) describes the practical aspects of this work. Chapter 4 describes the design and implementation of the **WSA** architecture in full custom 3μ CMOS VLSI. This chip, the LGCA chip, is the first CMOS chip to be realized at the Princeton University Department of Computer Science. It was found to operate at up to a 7 MHz clock rate, giving a maximum performance of 14 million lattice site updates per second.

Chapter 5 gives the details of the construction of a prototype lattice gas machine, LGM-1, based on the LGCA (lattice gas cellular automata) chips described in Chapter 4. The LGM-1 uses a pipeline of ten of the special purpose chips and the two board prototype is installed in a Sun 3/160 workstation host. The system interface is simple and has small bandwidth, so the LGM-1 can provide only 7 million site updates per second even though the chips are capable of 20 times more performance.

Nevertheless, the performance exceeds that of much larger machines of much greater cost. For example, the 32 node INTEL iPSC hypercube can perform 2.3 million site updates per second, an Alliant FX-8 vector processor can update 0.7 million sites per second, and the CAM-6 can perform 4 million site updates per second. The massively parallel Connection Machine, with its 65536 processors, sets the performance standard with 1 billion site updates per second [54].

The fact that our machine achieves high performance even though its hardware is slow

demonstrates the effectiveness of linear speedup. Unfortunately, our prototype machine has reached its performance limit because of the boundary value problem that we described earlier. Extensibility will require the design of a new chip or processor.

Chapter 6 describes a general host support system that should facilitate the construction of future linear systolic and pipelined special purpose VLSI processors. Its design is guided by the desire to make it easy to add hardware to extend the length of the special purpose pipeline.

7.2. Next Steps

One of the things that must be done immediately is to relax the 10 chip restriction on pipeline depth. As discussed earlier, the host has no access to data that is in the processor pipeline, so we are faced with the problem of not being able to extend the processor pipeline because we are unable to update the edge conditions frequently enough. If we were to add hardware to the pipeline to allow the pipeline itself to modify sites on the edges, then we could theoretically extend the pipeline indefinitely.

We propose to do this with a "boundary processor" that would be inserted into the processor pipeline at regular intervals. The interval should be adjustable, but should be no more than every 10 chips. Every time an edge site passes through the boundary processor, it will replace the value at that site with a random value that conforms to the probabilities given by Equation 1-1. This will allow a pipeline longer than 10 chips to be built because edge updates will take place in the pipeline as well as in the host.

In theory, the boundary processor could be placed on the same chip as the update processors. If they could be made small enough so that the amount shift register memory on chip would not be reduced, this would be an ideal situation. There would then be only one type of chip instead of the two (LGCA chip and Boundary Processor Chip) that we currently propose.

There are two main problems associated with the design of this processor. The first is the creation of weighted binary bits. A standard method is to create a word of pseudorandom bits using a linear feedback shift register [58] and compare this word to a threshold (the weight). If the random word exceeds the threshold, the result (weighted bit) is 0, otherwise it is 1. We have begun investigations into structures that would allow the parallel computation of many random bits from a pair of shift registers.

The second main problem is to distinguish an edge site from a non-edge site. One solution to this problem is to count the sites as they go through the pipeline. Since we know the dimensions of the lattice, we know exactly when the edge sites reach each stage of the pipeline. It is then an easy matter to substitute a random site value for the one currently in the pipeline. A small prototype that implements this strategy is being constructed by undergraduate Tarun Khanna [56].

A second possibility is to augment the state of a site from eight to nine bits by using additional

shift registers in parallel with the ones in the LGCA chip. The shift registers for the extra bit could be placed on the same chip as the pseudo random number generator and comparators, but could also be kept separate. Further thought will be given to this approach and simulations will have to be performed before we will know whether this technique is feasible.

Some prototyping environment such as that described in Chapter 6 should have a high priority. Without it, hardware development and experimentation is stalled. Any new chips (such as the boundary processor) or new hardware will require the design of new interface and support board hardware.

Finally, one of the most exciting prospects for future work is to extend our ideas to the development of a machine which simulates a *three* dimensional lattice gas cellular automaton. A lattice which meets the criteria for convergence to the full three dimensional Navier-Stokes equations has been published [31]. Graduate student Carl Staelin has done some preliminary work on the specification of particle collision rules, and on techniques for efficient simulation of the automaton [59]. This is a daunting problem. Each site in the three dimensional model has 24 neighbors, and there are thousands of collision rules. Currently, three dimensional fluid dynamics problems are solved by numerical integration on large supercomputers. Success in this endeavor could lead to an alternative technique for fluid dynamics computation. The three dimensional lattice gas automata also provide a valuable opportunity to explore highly parallel three dimensional lattice computations — presently beyond the capabilities of the largest computing machines.

References

1. G. Barnes, R. Brown, M. Kato, D. Kuck, D. Slotnick, and R. Stokes, "The ILLIAC IV Computer," *IEEE Transactions on Computers*, vol. C-17, no. 8, pp. 746-757, August 1968.
2. K. E. Batcher, "Design of a Massively Parallel Processor," *IEEE Transactions on Computers*, vol. C-29, no. 9, pp. 836-840, September 1980.
3. D. M. Nosenchuck and M. G. Littman, "The Coming Age of the Parallel-Processing Supercomputer," presented at the 23rd Annual Space Conference, John F. Kennedy Space Center, Florida, April 22, 1986.
4. S. Manohar and G. M. Baudet, "PE-I: A Super-coprocessor for solving PDE's," Technical Report No. CS-86-06, Brown University, Department of Computer Science, Providence, Rhode Island 02912, March 1986.
5. J. Von Neumann, *Theory of Self-Reproducing Systems*, University of Illinois Press, Urbana, 1966. ed. Arthur W. Burks
6. R. P. Feynman, "Simulating Physics with Computers," *International Journal of Theoretical Physics*, vol. 21, no. 6/7, 1982.
7. S. Wolfram, "Cellular automata as models of complexity," *Nature*, vol. 311, pp. 419-424, October 1984.
8. Cellular Automata 86, Cambridge, MA, June 16-18, 1986.
9. N. Margolus, "Physics-Like Models of Computation," in *Cellular Automata (Physica 10D)*, ed. S. Wolfram, pp. 81-95, North-Holland Physics Publishing, Amsterdam, 1984.
10. B. L. Buzbee, "Two Parallel Formulations of Particle-In-Cell Models," in *Algorithmically Specialized Parallel Computers*, ed. H. J. Siegel, pp. 223-231, Academic Press, 1985.
11. S. R. Sternberg, "Computer Architectures Specialized for Mathematical Morphology," in *Algorithmically Specialized Parallel Computers*, ed. H. J. Siegel, pp. 169-176, Academic Press, 1985.
12. T. Hatori and D. Montgomery, Transport Coefficients for Magnetohydrodynamic Cellular Automata, Department of Physics, Dartmouth College, January 1987. Unpublished Manuscript
13. J. K. Park, K. Steiglitz, and W. P. Thurston, "Soliton-Like Behavior in Automata," *Physica D*, vol. 19D, pp. 423-432, 1986. Reprinted in *Theory and Applications of Cellular Automata*, (S. Wolfram, ed.), World Scientific Publishing Co., Hong Kong, 1986, pp. 333-342.
14. K. Steiglitz, I. Kamal, and A. Watson, "Embedding Computation in One-Dimensional Automata by Phase Coding Solitons," *IEEE Trans. on Computers*, vol. 37, no. 2, pp. 138-145, Feb. 1988.
15. C. H. Goldberg, "Toward a Theory of Parity Filter Automata," Unpublished Manuscript, Dept. Comp. Sci., Princeton University, Princeton, NJ, November 12, 1987.
16. S. Harris, "Approach to Equilibrium in a Moderately Dense Discrete Velocity Gas," *The Physics of Fluids*, vol. 9, no. 7, pp. 1328-1332, 1966.
17. U. Frisch, B. Hasslacher, and Y. Pomeau, "A Lattice Gas Automaton For The Navier Stokes Equation," *Phys. Rev. Lett.*, vol. 56, no. 14, pp. 1505-1508, April 7, 1986.
18. M. Van Dyke, *An Album of Fluid Motion*, Parabolic Press, Stanford, CA, 1982.
19. J. R. Herring, S. A. Orszag, R. H. Kraichnan, and D. G. Fox, "Decay of Two-Dimensional Homogeneous Turbulence," *Journal of Fluid Mechanics*, vol. 66, no. 3, pp. 417-444, 1974.
20. S. A. Orszag, "Analytical Theories of Turbulence," *Journal of Fluid Mechanics*, vol. 41, no. 2, pp. 363-386, 1970.
21. P. J. Roache, *Computational Fluid Dynamics*, Hermosa Publishers, Alburquerque, New Mexico, 1972.
22. R. B. Pearson, J. L. Richardson, and D. Toussaint, "A Fast Processor for Monte-Carlo Simulation," *Journal of Computational Physics*, vol. 51, pp. 241-249, 1983.

23. A. Hoogland, J. Spaa, B. Selman, and A. Compagner, "A Special-Purpose Processor for the Monte Carlo Simulation of Ising Spin Systems," *Journal of Computational Physics*, vol. 51, pp. 250-160, 1983.

24. N. Margolus, T. Toffoli, and G. Vichniac, "Cellular-Automata Supercomputers for Fluid-Dynamics Modeling," *Physical Review Letters*, vol. 56, no. 16, pp. 1694-1696, April 21, 1986.

25. T. Toffoli and N. Margolus, *Cellular Automata Machines: A New Environment for Modeling*, MIT Press, Cambridge, MA, 1987.

26. G. Zannetti, personal correspondence..

27. A. L. Fisher and H. T. Kung, "Synchronizing Large VLSI Processor Arrays," *IEEE Transactions on Computers*, vol. c-34, no. 8, pp. 734-740, August 1985.

28. S. D. Kugelmass, R. Squier, and K. Steiglitz, "Performance of VLSI Engines for Lattice Computations," *Journal of Complex Systems*, vol. 1, no. 5, pp. 939-965, October 1987.

29. J. B. Salem and S. Wolfram, "Thermodynamics and Hydrodynamics with Cellular Automata," in *Theory and Applications of Cellular Automata*, ed. S. Wolfram, pp. 362-366, World Scientific Publishing Co., Hong Kong, 1986.

30. J. Hardy, Y. Pomeau, and O. de Pazzis, "Time evolution of a two-dimensional model system. I. Invariant states and time correlation functions," *J. Math. Phys.*, vol. 14, no. 12, pp. 1746-1759, December 1973.

31. D. d'Humiéres, P. Lallemand, and U. Frisch, "Lattice Gas Models for 3D Hydrodynamics," *Europhysics Letters*, vol. 4, no. 2, pp. 291-299, 15 August 1986.

32. S. A. Orszag and V. Yakhot, "Reynolds Number Scaling of Cellular Automaton Hydrodynamics," *Physical Review Letters*, vol. 56, no. 16, pp. 1691-1693, April 21, 1986.

33. P. A. Ruetz and R. W. Broderson, "Architectures and Design Techniques for Real-Time Image-Processing IC's," *IEEE Journal of Solid-State Circuits*, vol. SC-22, no. 2, April 1987.

34. J. Kittler and M. J. B. Duff, eds., *Image Processing System Architectures*, Research Studies Press, Ltd., John Wiley and Sons, Inc., 1985.

35. K. Steiglitz and R. R. Morita, "A Multi-Processor Cellular Automaton Chip," *Proc. 1985 IEEE Int. Conf. on Acoustics, Speech, and Signal Processing*, Tampa, Florida, March 1985.

36. A. L. Rosenberg, "Preserving Proximity in Arrays," *SIAM J. Computing*, vol. 4, no. 4, pp. 443-460, December 1975.

37. S. R. Sternberg, "Pipeline Architectures For Image Processing," in *Multicomputers and Image Processing, Algorithms and Programs*, ed. L. Uhr, pp. 291-305, Academic Press, 1982.

38. Kenneth Supowit and Neal Young, personal communication, 1986.

39. S. Y. Kung, S. C. Lo, S. N. Jean, and J. N. Hwang, "Wavefront Array Processors - Concept to Implementation," *Computer*, vol. 20, no. 7, IEEE, New York, July 1987.

40. R. L. M. Dang and N. Shigyo, "A two-dimensional simulation of LSI interconnect capacitance," *IEEE Electron Device Lett.*, vol. EDL-2, August 1981.

41. K. C. Saraswat and F. Mohammadi, "Effect of scaling of interconnections on the time delay of VLSI circuits," *IEEE J. Solid State Circuits*, vol. SC-17, no. 2, April 1982.

42. M. Hatamian, "Understanding Clock Skew in Synchronous Systems," *1987 Princeton Workshop on Algorithm, Architecture and Technology Issues in Models of Concurrent Computations*.

43. S. Ross, *A First Course in Probability*, Macmillan Publishing Company, New York, 1984.

44. W. Feller, *An Introduction to Probability Theory and Its Applications, Volume II*, John Wiley & Sons, New York, 1971.

45. H. Cramér, *Mathematical Methods of Statistics*, Princeton University Press, Princeton, NJ, 1946.

46. M. S. Paterson, W. L. Ruzzo, and L. Snyder, "Bounds on Minimax Edge Length for Complete Binary Trees," *Proc. Thirteenth Annual ACM Symp. Theory Comput.*, pp. 293-299, May 1981.

47. C. A. Mead and M. Rem, "Cost and Performance of VLSI Computing Structures," *IEEE J. Sold-State Circuits*, vol. SC-14, pp. 455-462, April 1979.

48. J. P. Singh, "Synchronization of Large VLSI Systems," Unpublished Manuscript, Dept. Electrical Engineering, Princeton University.

49. N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*, Addison-Wesley, 1985.

50. *VMEbus Specification Manual*, Motorola Semiconductor Products, Inc., February 1985. Revision C

51. P. Horowitz and W. Hill, *The Art of Electronics*, Cambridge University Press, 1980.

52. W. I. Fletcher, *An Engineering Approach to Digital Design*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1980.

53. M. D. Taylor, Report of Senior Independent Work, Department of Computer Science, Princeton University, January, 1988.

54. F. Hayot, M. Mandal, and P. Sadayappan, Implementation and Performance of a Binary Lattice Gas Algorithm on Parallel Processor Systems, Department of Physics, The Ohio State University, 1987. Submitted to J. of Comp. Phys.

55. W. D. Hillis, "The Connection Machine: A Computer Architecture Based on Cellular Automata," in *Physica 10D*, pp. 213-228, North-Holland, Amsterdam, 1984.

56. T. Khanna, Report of Senior Independent Work, Department of Computer Science, Princeton University, January, 1988.

57. R. J. Lipton and D. Lopresti, "Comparing Long Strings on a Short Systolic Array," TR CS-026, Princeton University, Princeton, NJ, February 12, 1986.

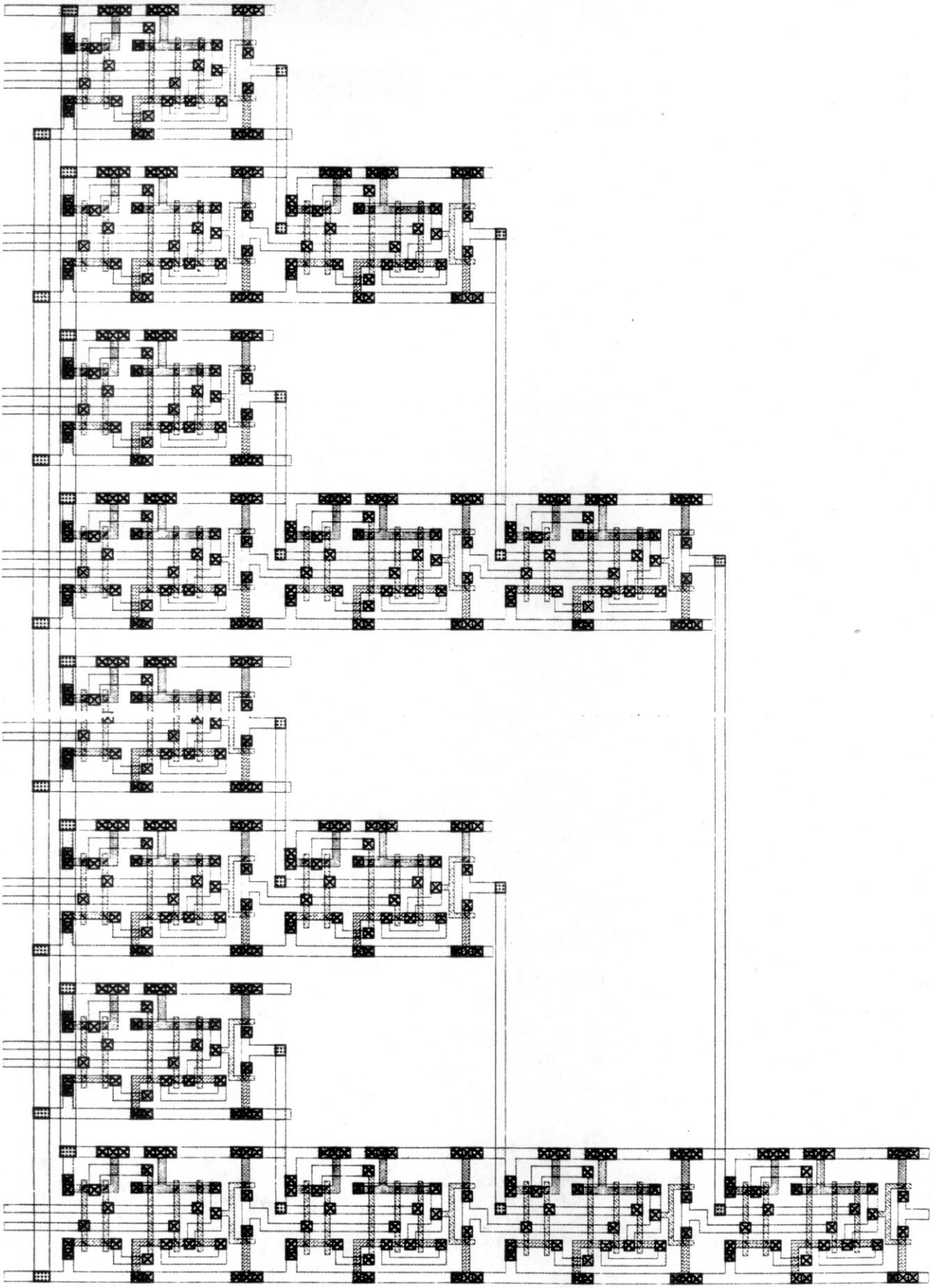
58. S. W. Golomb, *Shift Register Sequences*, Holden-Day, Inc., San Francisco, 1967.

59. Carl Staelin, private communication, 1987.

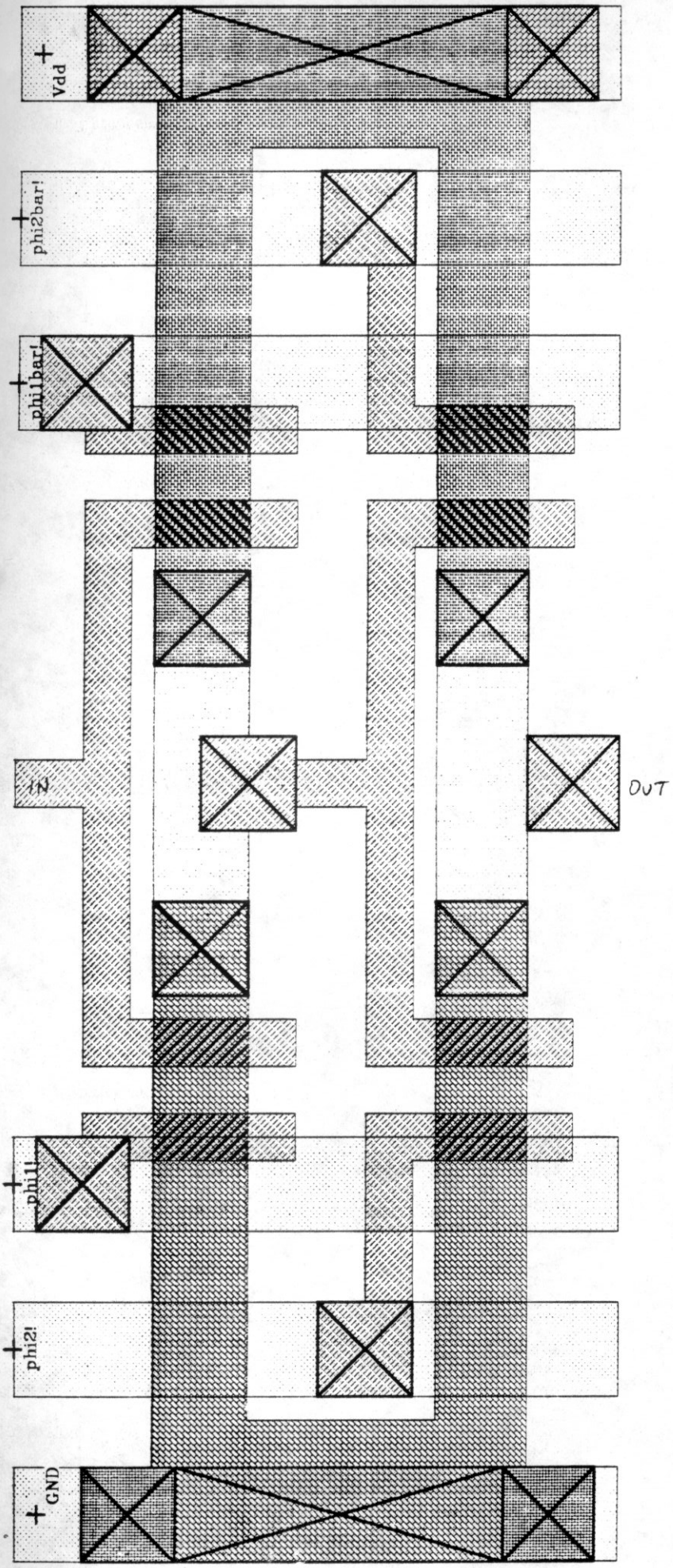
Appendix 1

SFM-4 Pin Assignments	
1	Substrate
2	Data Out 07
3	Parity Out
4	
5	
6	
7	
8	
9	
10	
11	
12	
13	Phi 2 BAR
14	Phi 2 BAR
15	Phi 2
16	Phi 2
17	Vdd
18	Phi 1
19	Phi 1
20	Phi 1 BAR
21	Phi 1 BAR
22	
23	
24	
25	
26	
27	
28	
29	
30	
31	Control 1
32	Control 0
33	Data In 15
34	Data In 14
35	Data In 13
36	Data In 12
37	Data In 11
38	Data In 10
39	Data In 09
40	Data In 08
41	Data In 07
42	Data In 06
43	Data In 05
44	Data In 04
45	Data In 03
46	Data In 02
47	Data In 01
48	Data In 00
49	GND
50	Data Out 08
51	Data Out 09
52	Data Out 10
53	Data Out 11

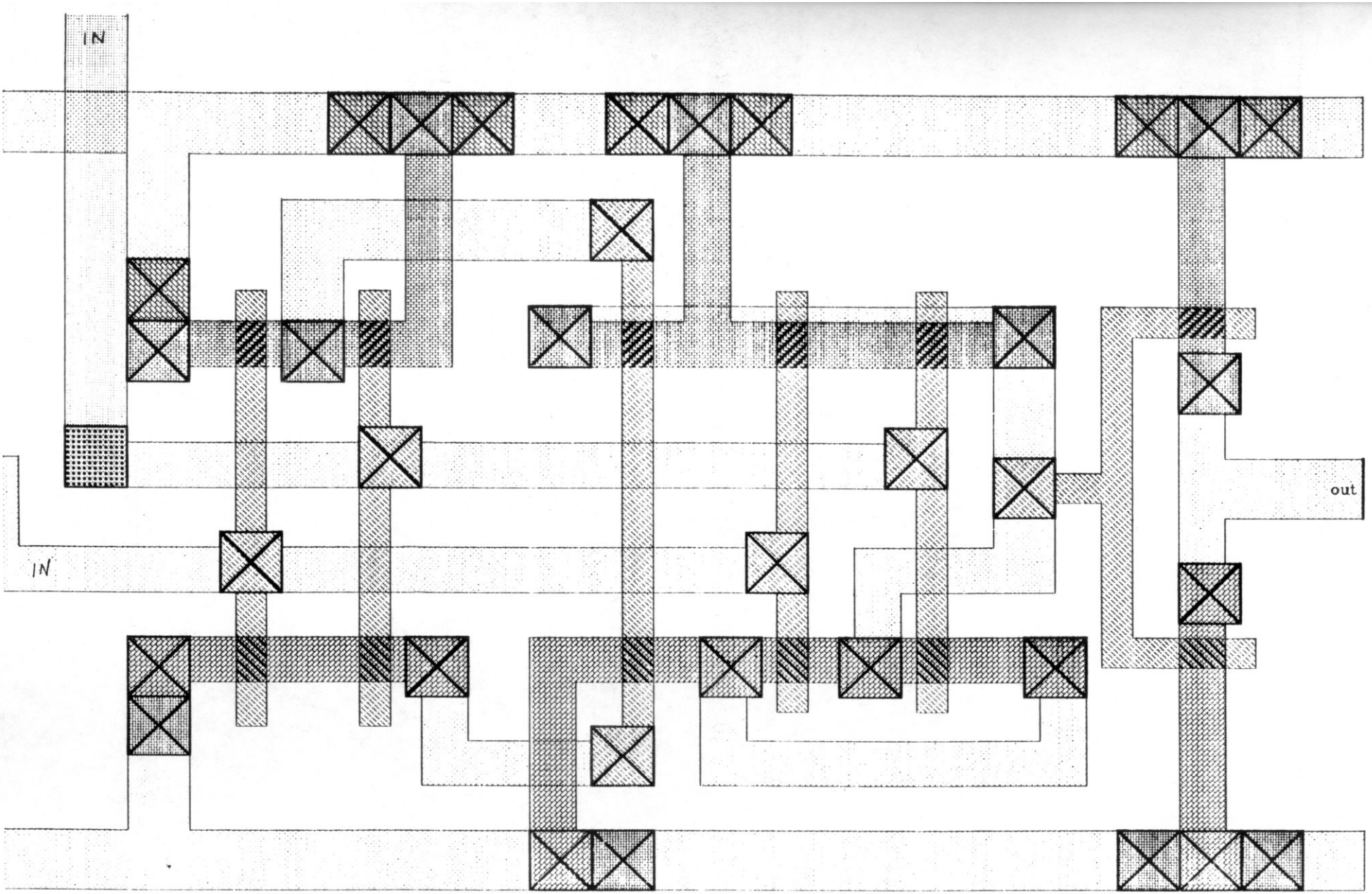
SFM-4 Pin Assignments	
54	Data Out 12
55	Data Out 13
56	Data Out 14
57	Data Out 15
58	Data Out 00
59	Data Out 01
60	Data Out 02
61	Data Out 03
62	Data Out 04
63	Data Out 05
64	Data Out 06



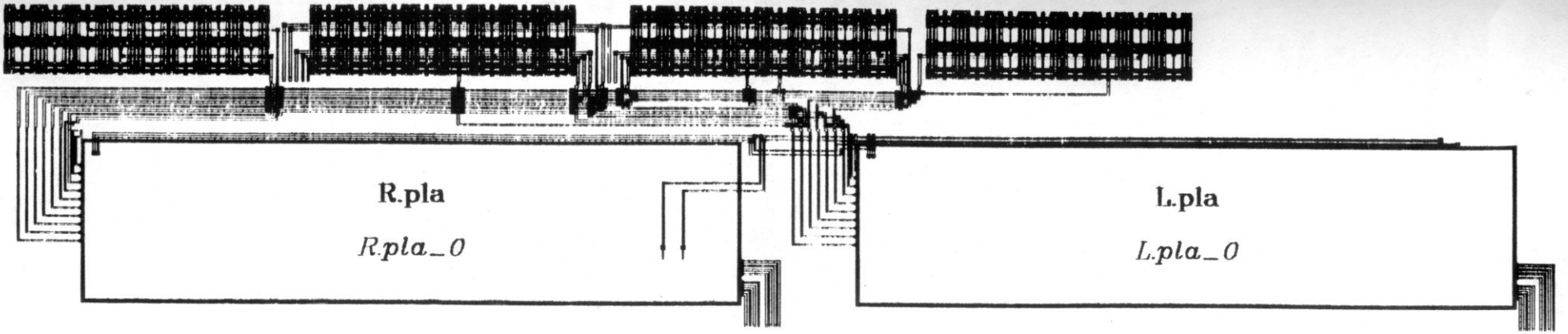
XOR TREE



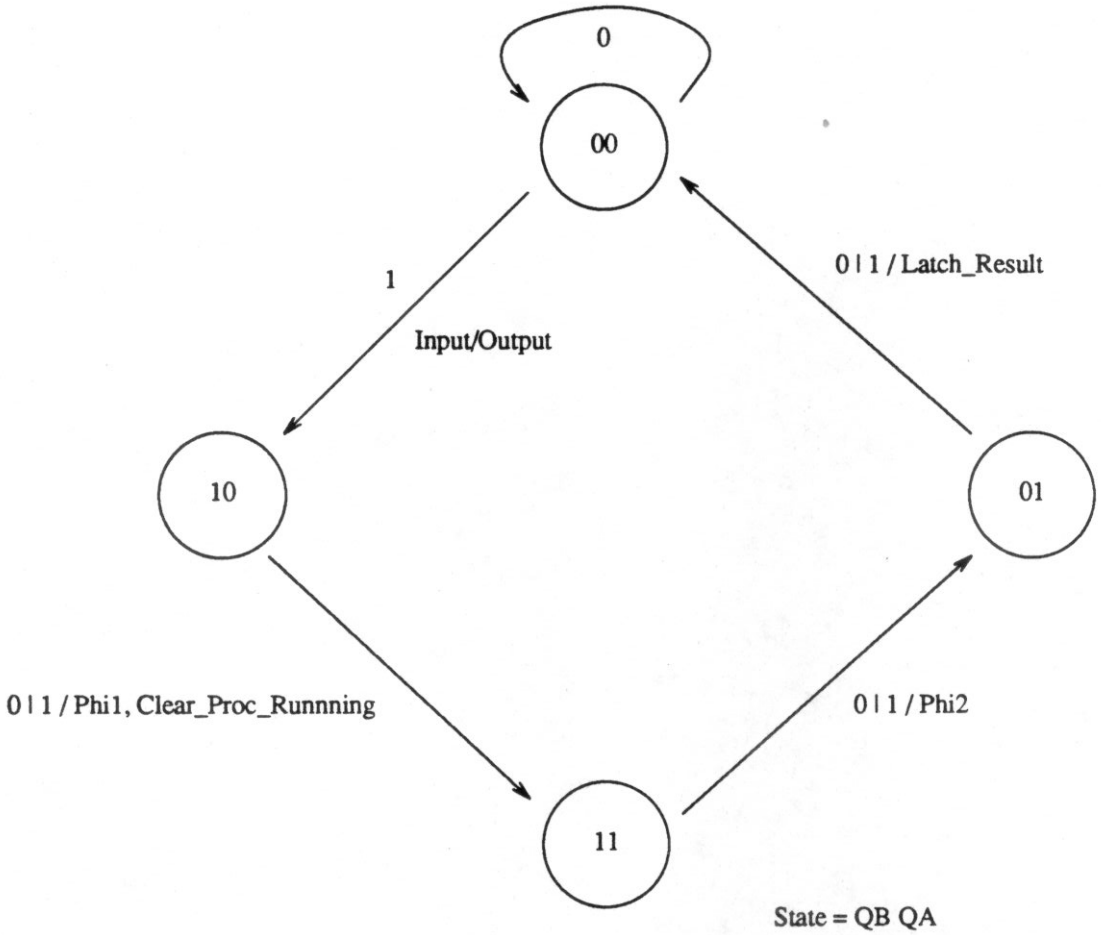
SHIFT REGISTER
CELL



XOR GATE


R.pla
R.pla_0
L.pla
L.pla_0

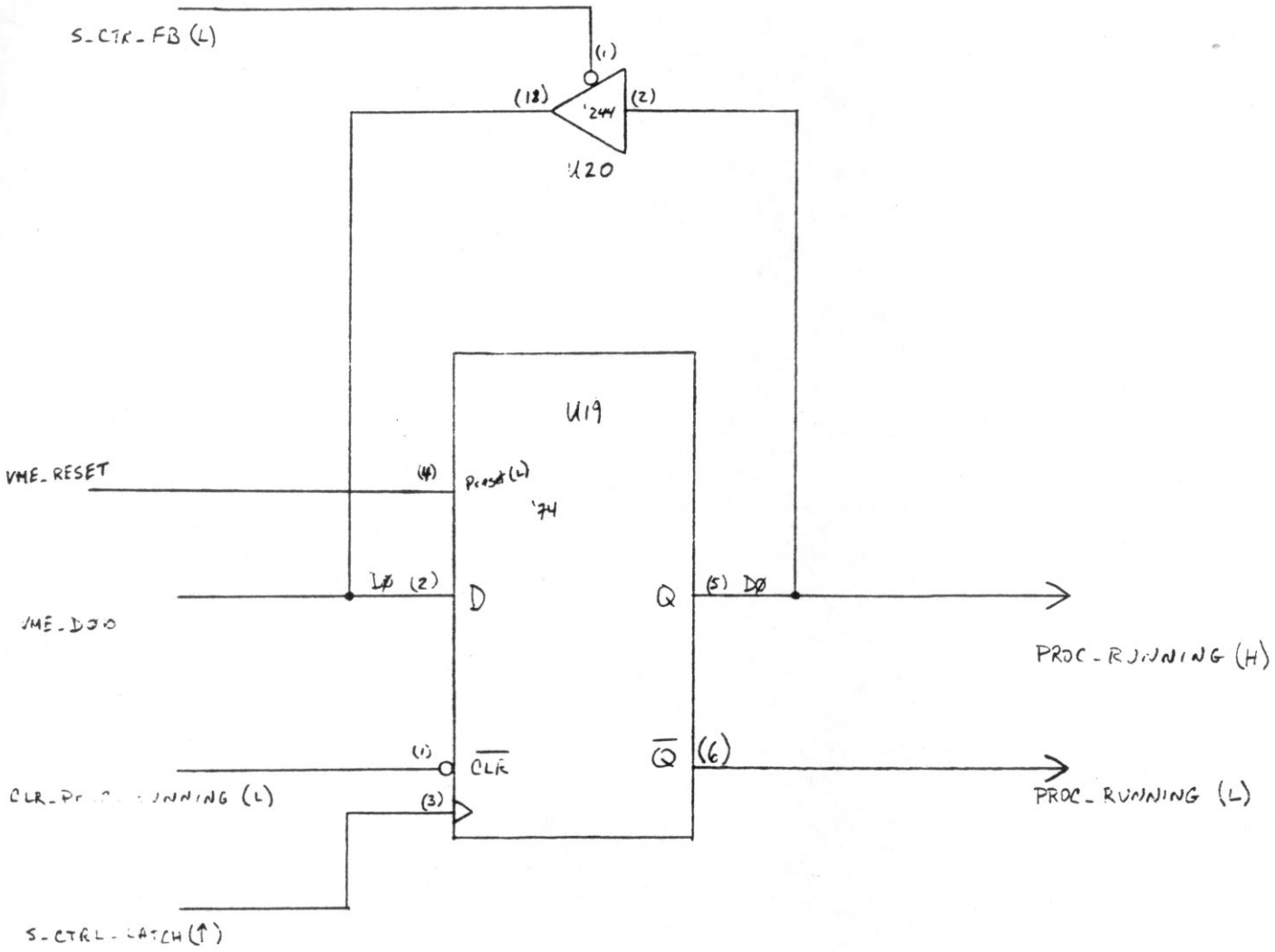
Clock State Machine

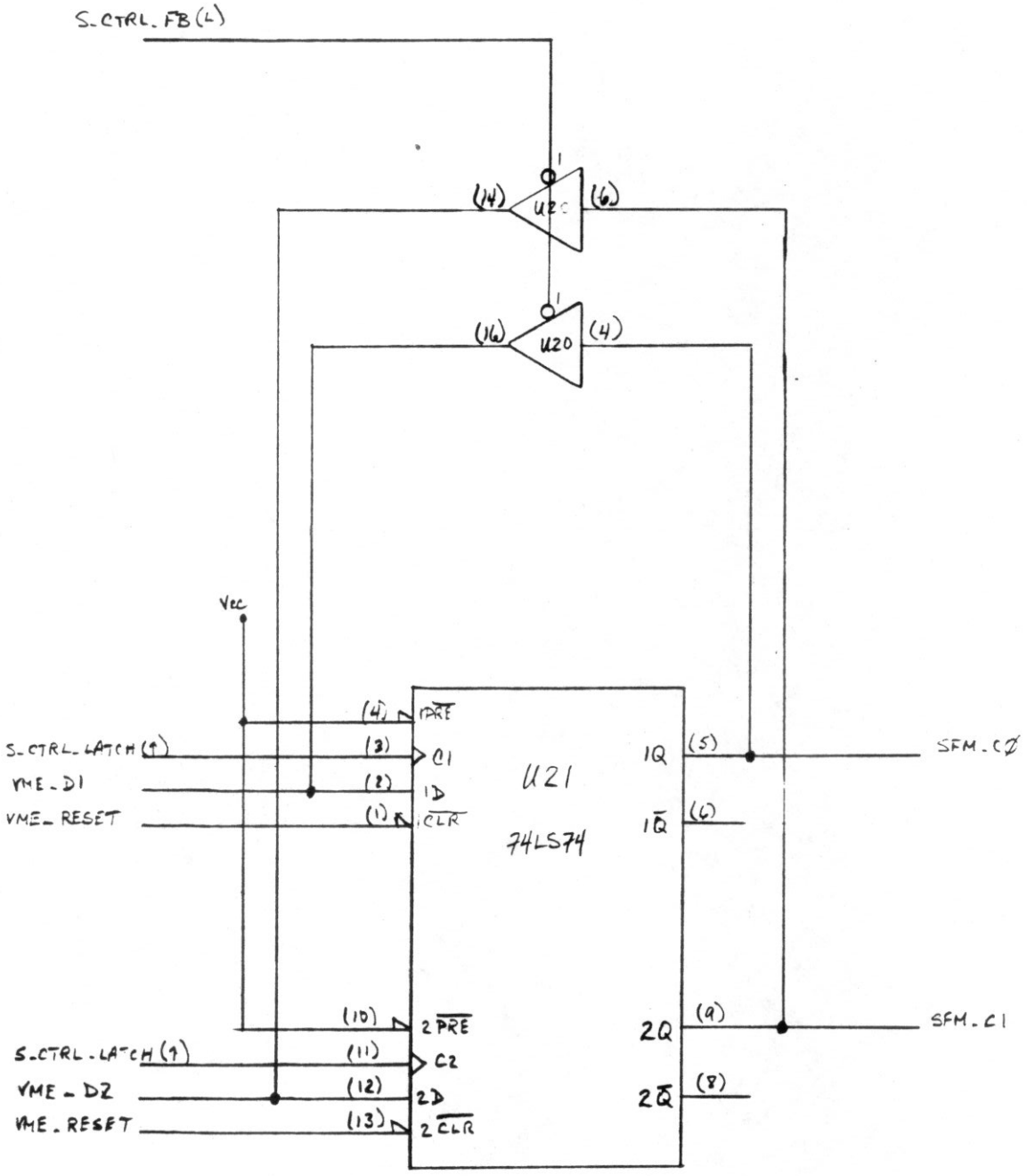


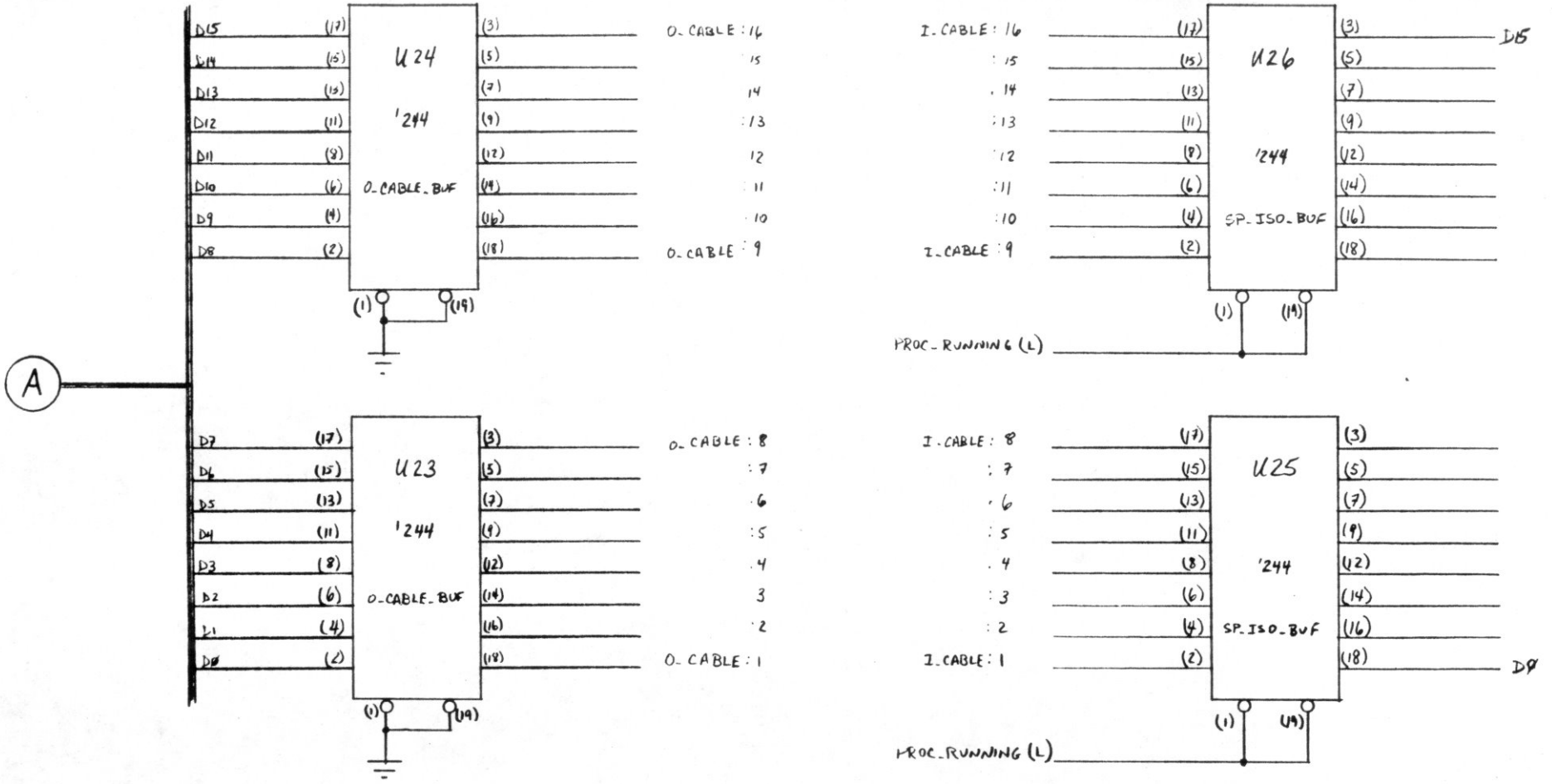
State = QB QA

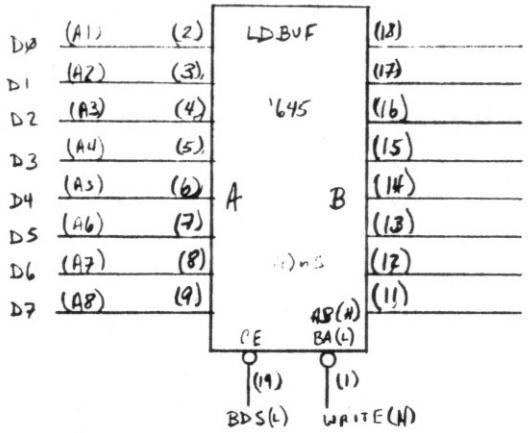
Input = Proc_Running

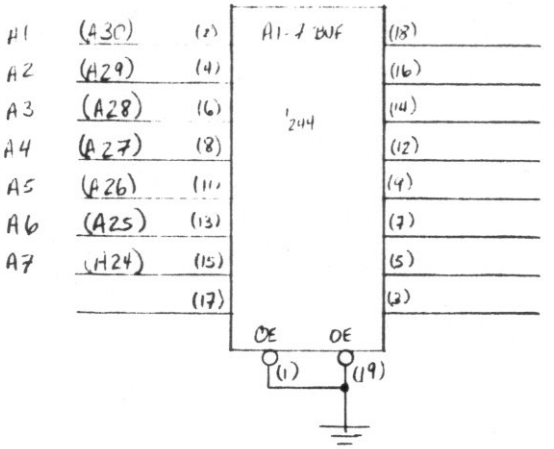
Ouput = Phi1, Phi2, Clear_Proc_Running

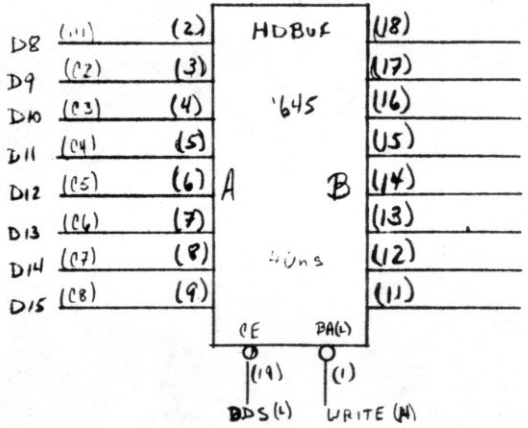

S_CTR_FB (L)
(1)
(18)
'244
(2)
U20
U19
VME_RESET
(4)
Preset (L)
'74
D0 (2)
D
Q
(5) D0
VME_D00
PROC_RUNNING (H)
(1)
CLR
Q
(6)
CLR_PROC_RUNNING (L)
PROC_RUNNING (L)
(3)
S_CTRL_LATCH (↑)

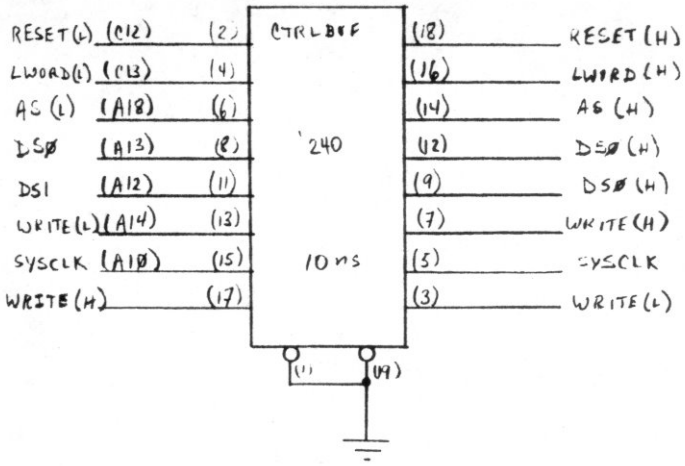

S_CTRL_FB (L)
U22
(14)
(6)
U20
(16)
(4)
Vcc
U21
74LS74
(4) 1PRE
S_CTRL_LATCH (↑) (3) 1C1
VME_D1 (2) 1D
VME_RESET (1) 1CLR
1Q (5)
1Q̄ (6)
SFM_C0
(10) 2PRE
S_CTRL_LATCH (↑) (11) 2C2
VME_D2 (12) 2D
VME_RESET (13) 2CLR
2Q (9)
2Q̄ (8)
SFM_C1


A
D15
D14
D13
D12
D11
D10
D9
D8
U24
'244
O_CABLE_BUF
O_CABLE : 16
: 15
: 14
: 13
: 12
: 11
: 10
O_CABLE : 9
D7
D6
D5
D4
D3
D2
D1
D0
U23
'244
O_CABLE_BUF
O_CABLE : 8
: 7
: 6
: 5
: 4
: 3
: 2
O_CABLE : 1
I_CABLE : 16
: 15
: 14
: 13
: 12
: 11
: 10
I_CABLE : 9
U26
'244
SP_ISO_BUF
PROC_RUNNING (L)
D15
I_CABLE : 8
: 7
: 6
: 5
: 4
: 3
: 2
I_CABLE : 1
U25
'244
SP_ISO_BUF
PROC_RUNNING (L)
D0

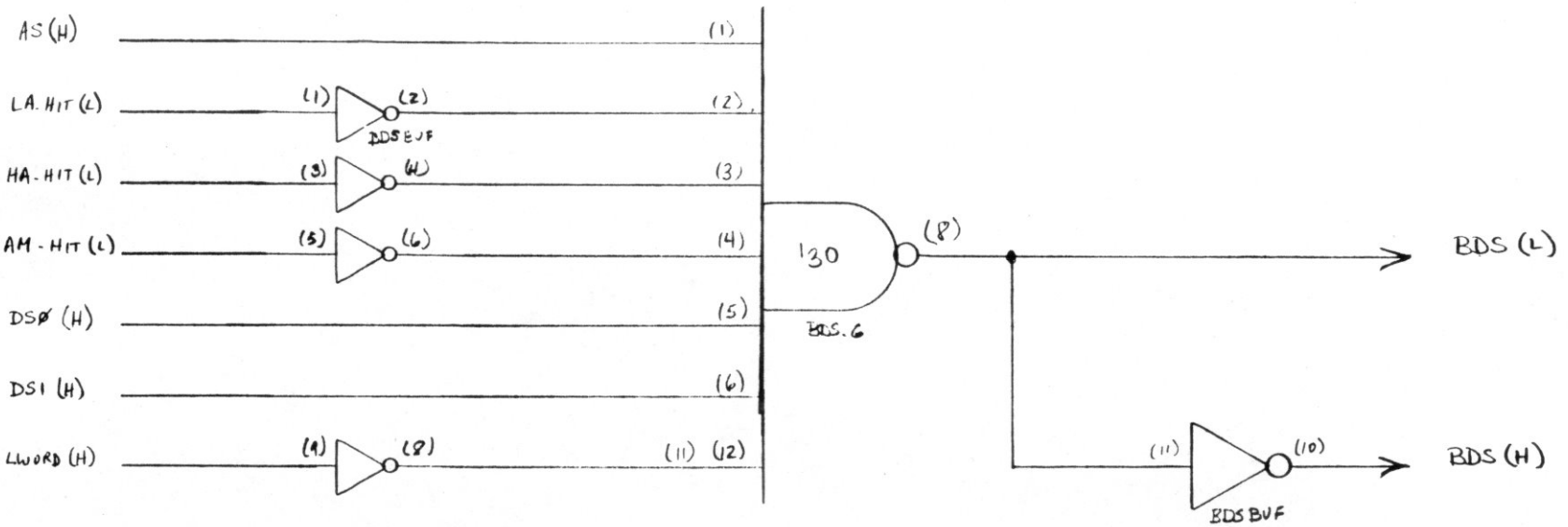




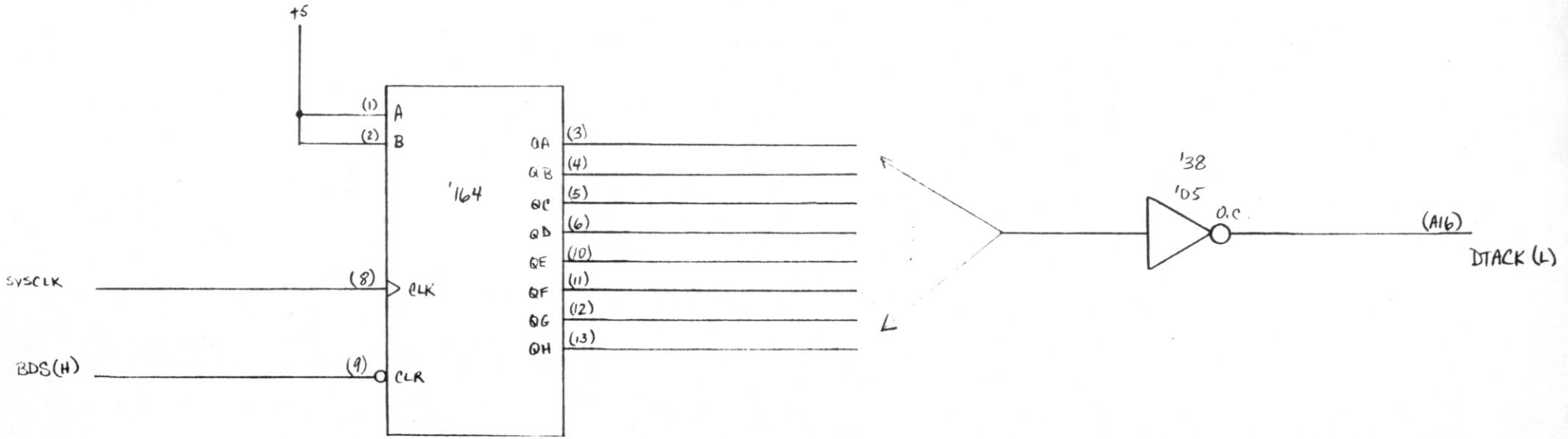




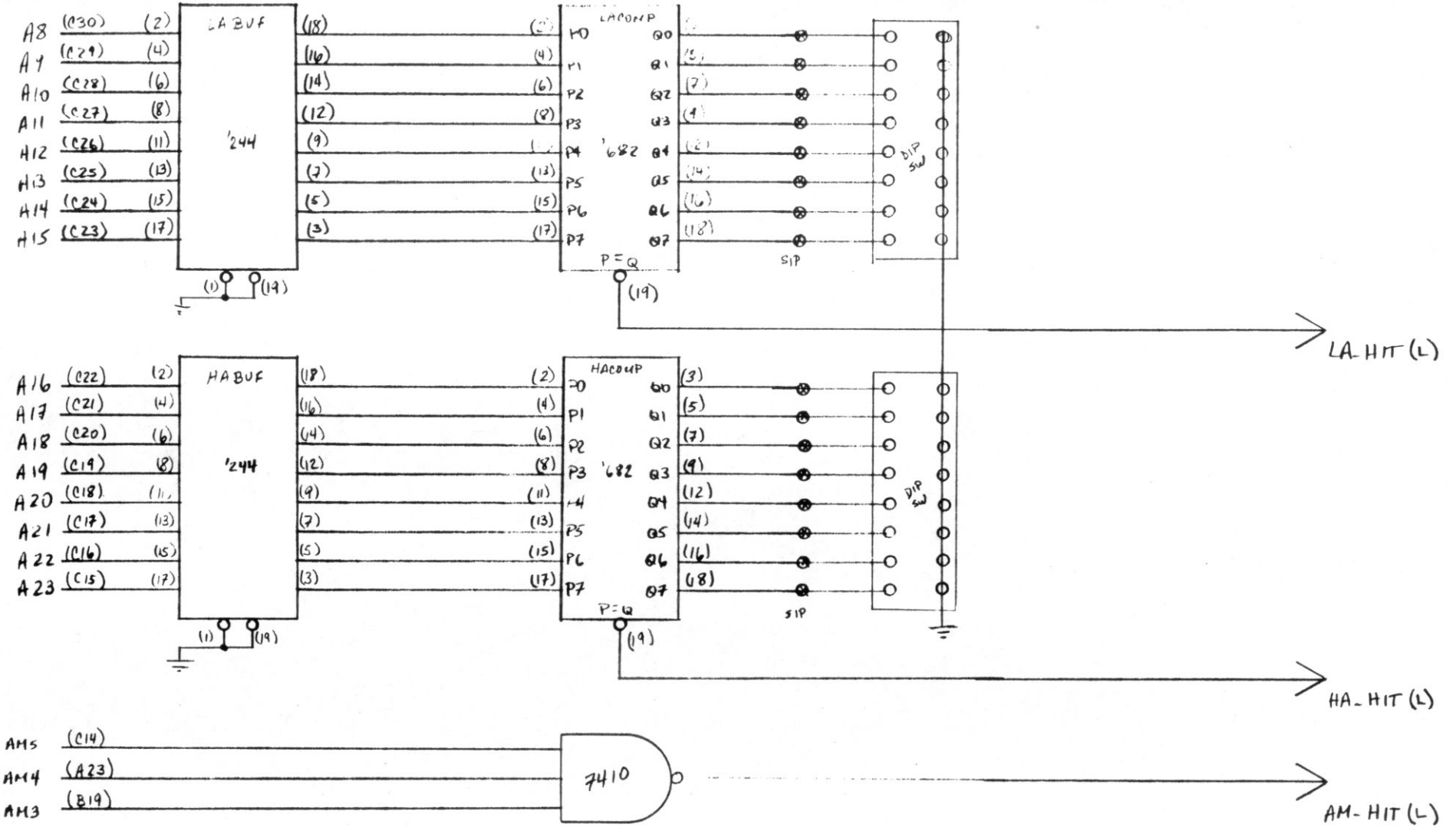
DATA BUFFERS
CONTROL SIGNAL BUFFERS
ADDRESS BUFFERS



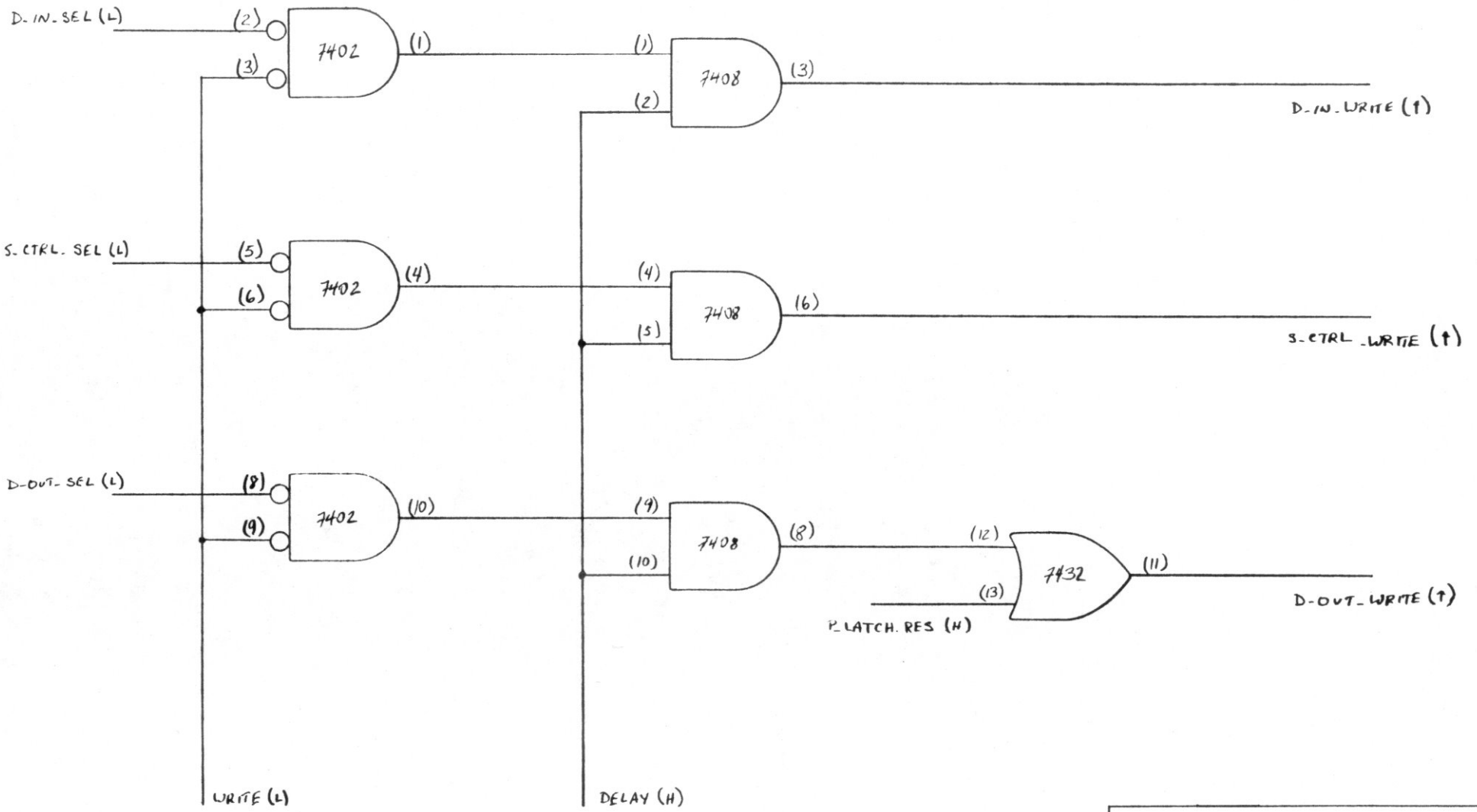
BOARD SELECTION



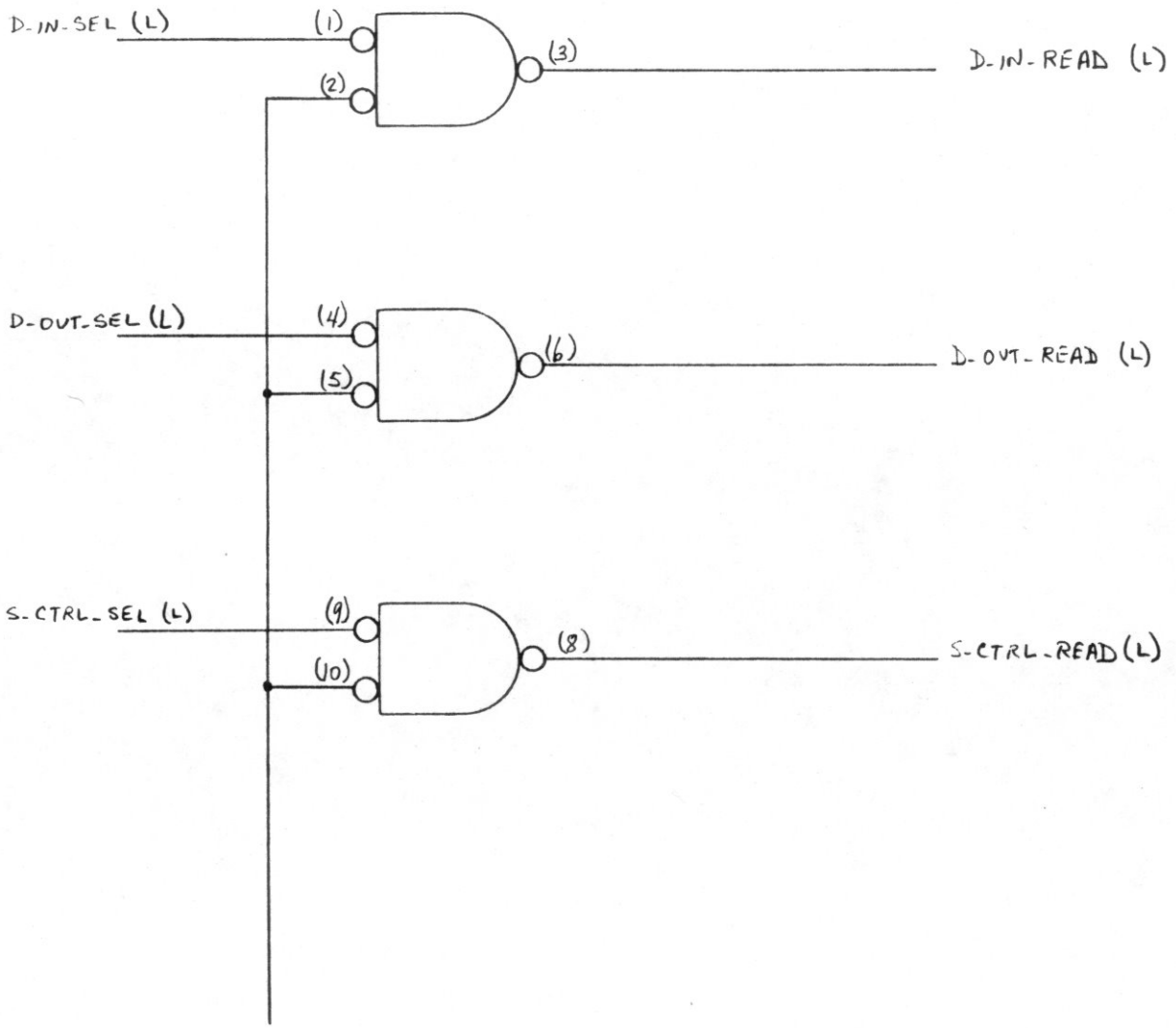
DTACK GENERATION



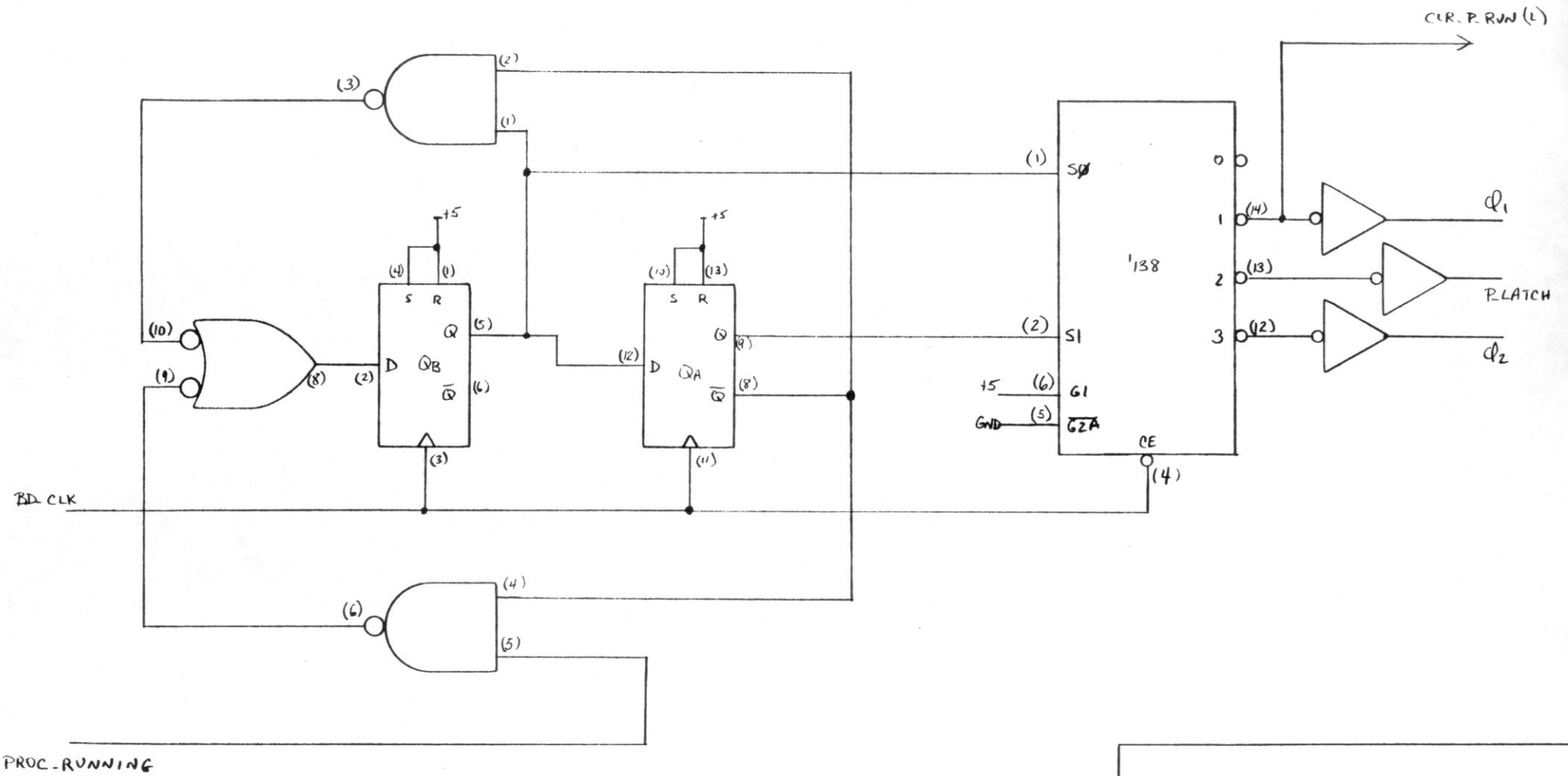
ADDRESS DETECTION



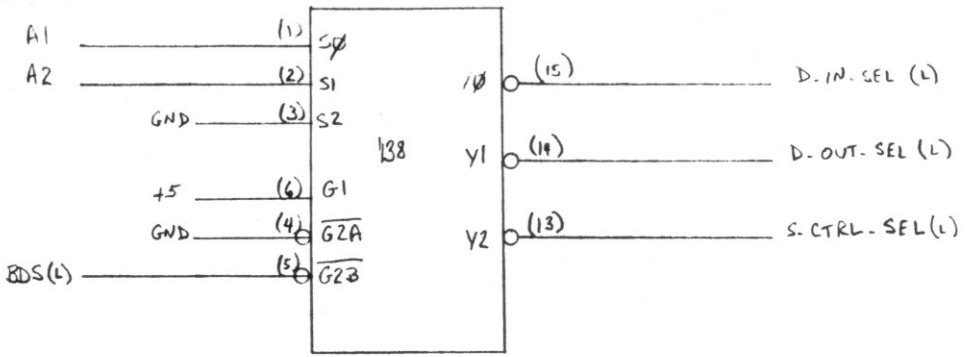
LATCH WRITE SIGNALS

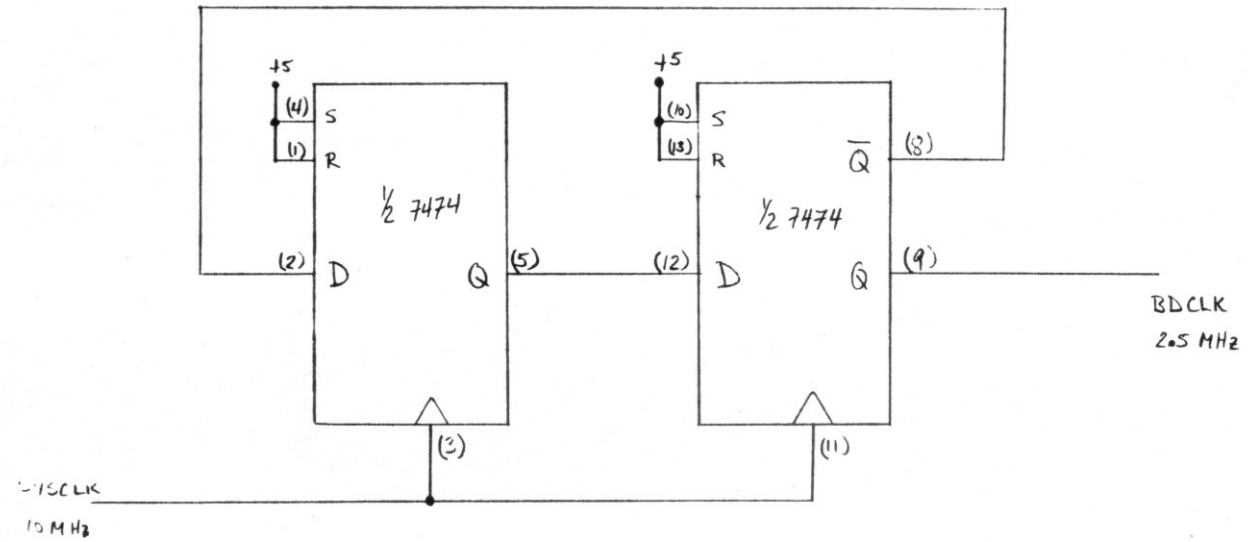


REGISTER READ SIGNALS

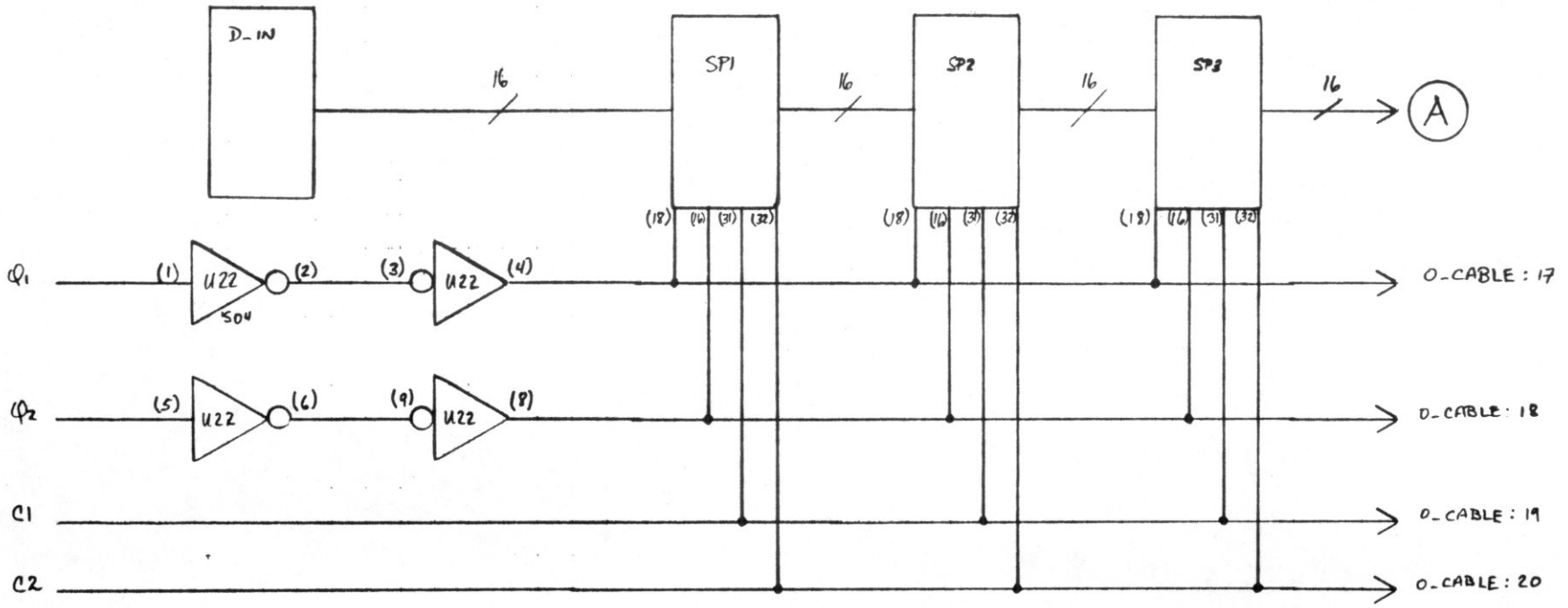


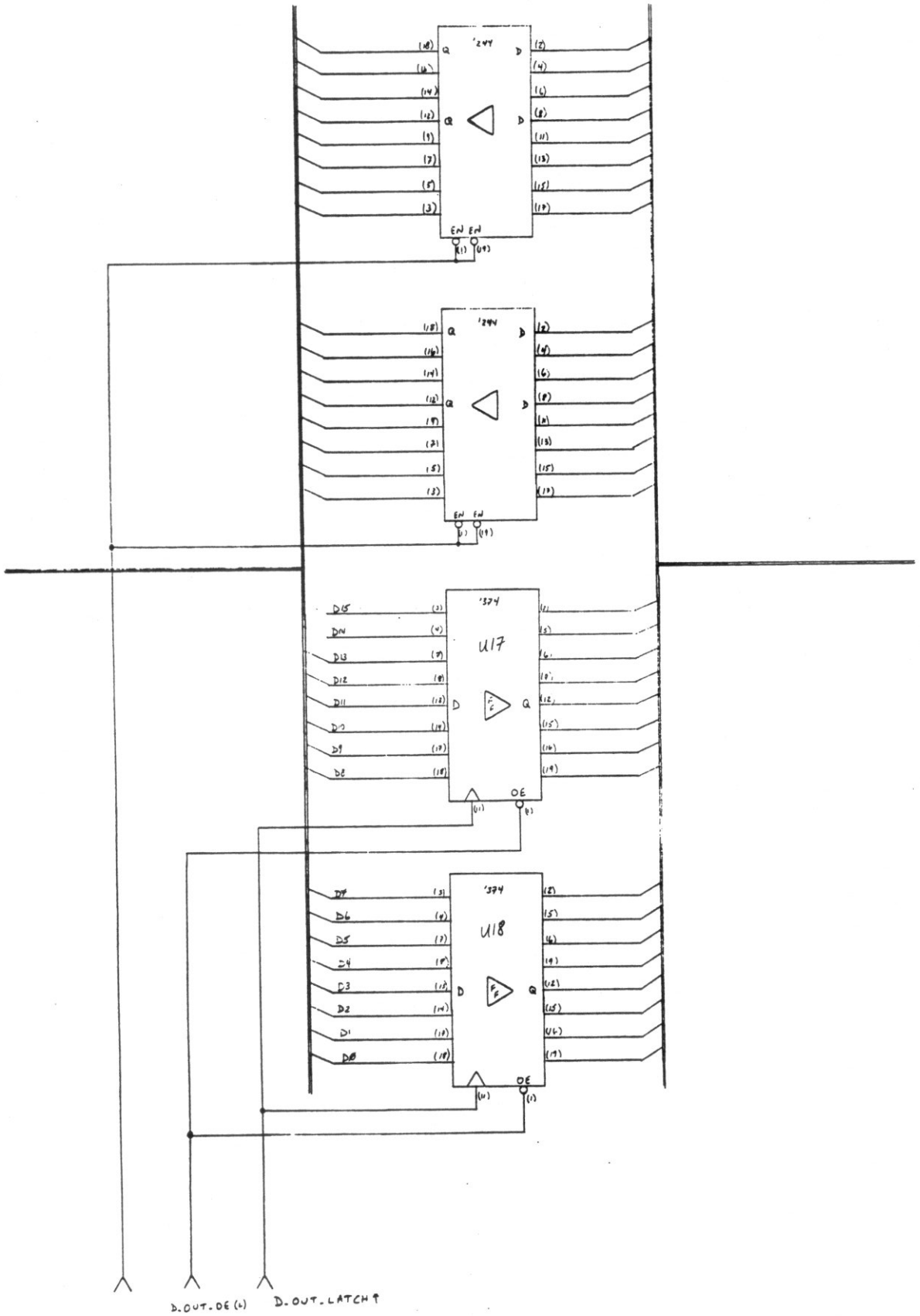
Array Control



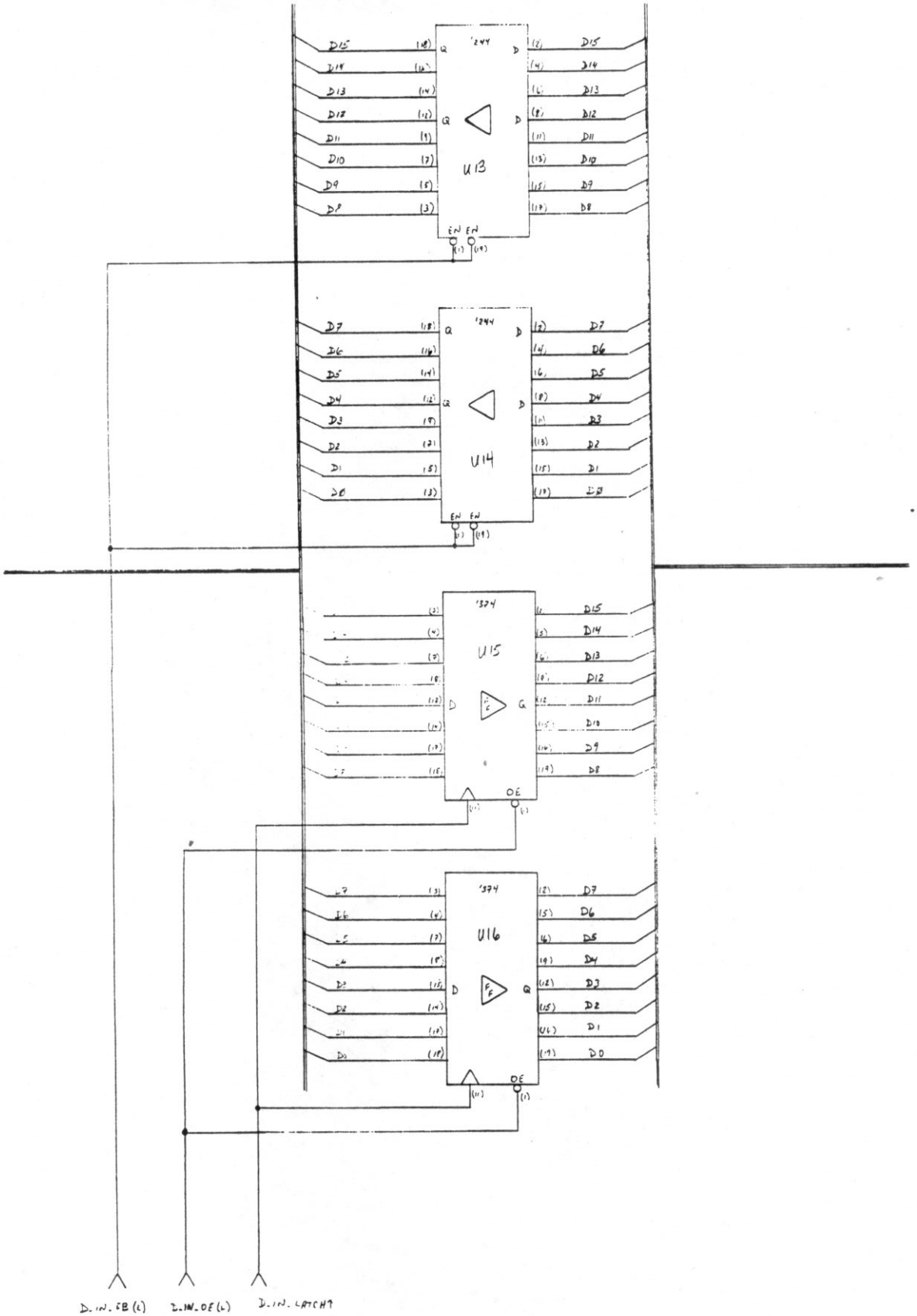


Address Decoding and
On-board Clock Generation


D_IN
16
SP1
16
SP2
16
SP3
16
A
(18)
(14)
(31)
(32)
(18)
(14)
(31)
(32)
(18)
(14)
(31)
(32)
Q1
(1)
U22
(2)
(3)
U22
(4)
SO4
Q2
(5)
U22
(6)
(9)
U22
(8)
C1
C2
O_CABLE : 17
O_CABLE : 18
O_CABLE : 19
O_CABLE : 20



Note: Data-Out Register has no feedback function.


'244
U13
D15
D14
D13
D12
D11
D10
D9
D8
EN EN
'244
U14
D7
D6
D5
D4
D3
D2
D1
D0
EN EN
'374
U15
D15
D14
D13
D12
D11
D10
D9
D8
D
Q
OE
'374
U16
D7
D6
D5
D4
D3
D2
D1
D0
D
Q
OE
D_IN_OB(L)
D_IN_OE(L)
D_IN_LATCH