

---

Retrospective Theses and Dissertations

---

1987

## A Reconfigurable Orthogonal Systolic Array Implementation of a Kalman Filter

Mark V. Bapst  
*University of Central Florida*

 Part of the [Engineering Commons](#)

Find similar works at: <https://stars.library.ucf.edu/rtd>

University of Central Florida Libraries <http://library.ucf.edu>

This Masters Thesis (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Retrospective Theses and Dissertations by an authorized administrator of STARS. For more information, please contact [STARS@ucf.edu](mailto:STARS@ucf.edu).

---

### STARS Citation

Bapst, Mark V., "A Reconfigurable Orthogonal Systolic Array Implementation of a Kalman Filter" (1987). *Retrospective Theses and Dissertations*. 5087.

<https://stars.library.ucf.edu/rtd/5087>

UNIVERSITY OF CENTRAL FLORIDA

OFFICE OF GRADUATE STUDIES

THESIS APPROVAL

DATE: November 18, 1987

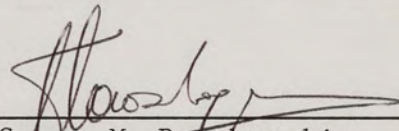
I HEREBY RECOMMEND THAT THE THESIS PREPARED UNDER MY SUPERVISION

BY Mark V. Bapst

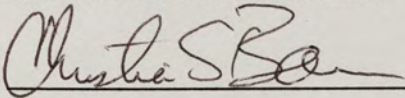
ENTITLED "A Reconfigurable Orthogonal Systolic Array  
Implementation of a Kalman Filter"

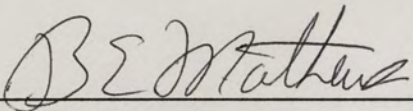
BE ACCEPTED IN PARTIAL FULFILLMENT OF THE REQUIREMENTS OF THE  
DEGREE OF Master of Science in Engineering

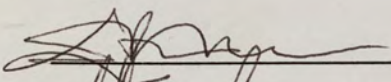
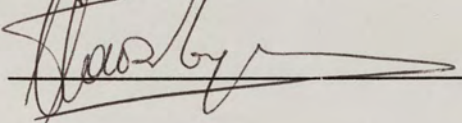
FROM THE COLLEGE OF Engineering

  
\_\_\_\_\_  
George M. Papadourakis  
Supervisor of Thesis

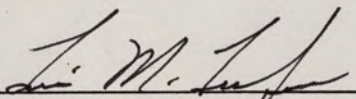
RECOMMENDATION CONCURRED IN:

  
\_\_\_\_\_

  
\_\_\_\_\_  
Bruce E. Mathews  
Coordinator of Degree Program

  
\_\_\_\_\_  
  
\_\_\_\_\_

COMMITTEE ON FINAL EXAMINATION

  
\_\_\_\_\_  
Louis M. Trefonas  
Dean of Graduate Studies

A RECONFIGURABLE ORTHOGONAL SYSTOLIC ARRAY  
IMPLEMENTATION OF A KALMAN FILTER

BY

MARK V. BAPST  
B.S.E., University of Florida, 1985

THESIS

Submitted in partial fulfillment of the requirements  
for the degree of Master of Science in Engineering  
in the Graduate Studies Program of the  
College of Engineering  
University of Central Florida  
Orlando, Florida

Fall Term  
1987



## ABSTRACT

An important part of optimal estimation technology, the Kalman filter is a computationally intensive application that has been limited either to non-real time realizations or to realizations that can afford vast amounts of mainframe hardware. The potential use of the Kalman filter theory could be greatly enhanced by a low cost, high performance machine capable of computing the recursive matrix equations in real time.

The use of pipelined parallel architectures allows the Kalman filter equations to be realized with much greater efficiency than previous implementations. A reconfigurable, few instruction, multiple data, orthogonal, pipelined, systolic array processor will be used to implement the recursive algorithm of the filter. Since the architecture is reconfigurable, a single systolic array will perform all of the required operations. The architecture selected provides a general foundation for other applications involving matrix computations to build upon.

A previously designed algorithm for pipelined matrix multiplication is employed, and a modified version of an inversion algorithm which is based on Cholesky's method is used. The resulting system improves the performance of the Kalman filter by about a factor of three over an implementation by Liu and Young.



## ACKNOWLEDGMENTS

The author wishes to thank his committee chairman, Dr. George M. Papadourakis, for his intellectual guidance in the development of the body of this report. The success of this project would not have been possible without his leadership.

The participation of thesis committee members Dr. Christian S. Bauer and Dr. Harley Myler is greatly appreciated. Also, Tina Andre verified the inverse algorithm, and her help is acknowledged.

Lastly, Donna M. Bapst has provided great motivation and encouragement during the duration of this project.

## TABLE OF CONTENTS

LIST OF TABLES . . . . .	iv
LIST OF FIGURES . . . . .	v
CHAPTER I, INTRODUCTION . . . . .	1
The Kalman Filter . . . . .	2
The Systolic Architecture . . . . .	3
The Logarithmic Number System . . . . .	5
CHAPTER II, DESIGN FUNDAMENTALS . . . . .	8
Kalman Filter Equations . . . . .	9
Systolic Architectures . . . . .	11
Arithmetic Algorithms For Systolic Arrays . . . . .	18
Matrix Load . . . . .	20
Matrix Multiplication . . . . .	22
Matrix Inversion . . . . .	25
Procedure For Upper Matrix Decomposition . . . . .	26
Procedure For Inversion Of Upper Matrix . . . . .	28
Example of Inversion Procedure . . . . .	31
Matrix Transposition . . . . .	33
PE Control . . . . .	34
Processing Element . . . . .	38
The LNS ALU . . . . .	39
LNS Arithmetic . . . . .	39
ALU Functional Description . . . . .	45
Scratch Pad Memory . . . . .	47
PE Control Structure . . . . .	47
The PE Microcontroller . . . . .	48
The Control Word . . . . .	49
CHAPTER III, DESIGN ANALYSIS . . . . .	51



Control of Successive Operations . . . . .	51
Order of Pipelined Operations . . . . .	52
Register Level Simulation . . . . .	55
Performance Comparison . . . . .	59
CHAPTER IV, CONCLUSION . . . . .	63
Alternate Architectures . . . . .	64
Areas of Future Work . . . . .	64
REFERENCES . . . . .	66



## LIST OF TABLES

1. Kalman Filter Variables . . . . .	10
2. Geometric Configurations and Corresponding Functions . . . . .	17
3. Memory Requirements for Logarithmic Addition . . . . .	44
4. Opcode Definition for ALU . . . . .	45
5. Opcode Definition For Internal Data Multiplexers . . . . .	46
6. Opcode Definition For PE Control . . . . .	49
7. Kalman Filter Operations Versus Time . . . . .	53
8. Kalman Filter Execution Time Comparison . . . . .	60
9. Comparison of Kalman Filter Implementation Versus N . . . . .	60
10. PE Utilization of Kalman Filter Implementation . . . . .	61

## LIST OF FIGURES

1. Models For Four Computer Classes . . . . .	13
2. Conventional Versus Systolic Processor Architectures . . . . .	14
3. Common Systolic Architecture Configurations . . . . .	16
4. Systolic Pipelined Orthogonal Array . . . . .	19
5. Matrix Load Procedure . . . . .	21
6. Matrix Multiply Procedure . . . . .	24
7. Upper Triangular Matrix Decomposition Procedure . . . . .	27
8. Upper Triangular Matrix Inversion Procedure . . . . .	29
9. Pipelined Inversion Procedure . . . . .	30
10. Block Diagram of Transpose Switch . . . . .	35
11. Control Structures For Orthogonal Array Algorithms . . . . .	37
12. Block Diagram of Processing Element . . . . .	40
13. Block Diagram of LNS ALU . . . . .	41
14. Data Flow For Decomposition and Inverse Operations . . . . .	57
15. Data Flow For Multiplication Operation . . . . .	58



## CHAPTER I, INTRODUCTION

Since its inception in 1960, the Kalman filter has been an important part of optimal filter technology. It has been introduced to a wide range of applications including missile guidance, air and sea navigation, target tracking, and flight control. The theory is ahead of practice. The filter's implementation has been hindered by the fact that it is computationally bound. The recursive nature of the Kalman filter coupled with the matrix equations used poses severe performance limitations on this technique for optimal linear estimation. An inexpensive yet powerful processor of compact proportions would broaden the filter's applications to include process control, robotics, and computer vision (Graham and Kadela 1985).

Advances in parallel computer architectures have provided promise for the reduction of the computational bottleneck associated with recursive, linear applications. However, these architectures would not be feasible without the tremendous progress in microelectronics. Very large scale integration (VLSI), the fourth generation of integrated circuits, has provided logic designers with the means to improve performance and reduce size of existing systems. More importantly, it has allowed them to set new state of the art design goals which is clearly evident in the 16 million transistor memories and 250,000 transistor microprocessors being developed today (Pucknell and Eshraghian 1985).



## The Kalman Filter

The intent of this paper is not simply to develop a specific processor that implements the Kalman filter in an efficient manner. Rather, the Kalman filter is presented to provide essentially a worst case computational application that will allow a systolic architecture to be developed that will perform matrix addition, subtraction, multiplication, inversion, and transposition. Since this application requires these five basic matrix operations, it provides a general platform that can be modified to perform most any recursive set of matrix computations.

The Kalman filter is used to estimate the state variables of a system when noise is present. The Kalman filter is the most popular state estimator used in system control in the least square sense. It is easily extended to nonlinear systems and systems with non-Gaussian noise (Graham and Kadela 1985).

Unfortunately, the intensive matrix calculations required to implement the Kalman filter have limited its applications, especially those applications requiring real time performance. Graham and Kadela considered the use of a systolic architecture to solve the Kalman filter dilemma. They restructured the optimal state estimation equations developed by Kalman to minimize the amount of processing necessary and to take advantage of the single instruction multiple data (SIMD) architectures afforded by systolic arrays. However, they did not employ any pipelining nor did they provide an arithmetic logic unit with the dynamic precision necessary in most optimal state estimation applications. This paper will modify existing algorithms for matrix

multiplication and inversion to take advantage of pipelining to improve performance. In addition, the systolic machine developed will have a high dynamic precision for realization in practical examples. This paper will go one step further in improving performance in implementing the Kalman filter machine.

### The Systolic Architecture

Over the past thirty years, computation speed has been increased primarily as a result of improved electronic technology. Integrated circuits have become faster and smaller as transistor feature size has been reduced. However, it has become evident that the technological advancements in integrated circuit technology have slowed due to complex quantum physics problems associated with reduced transistor minimum feature size. Since the technological trend indicates slower improvements in component speed, designers must consider other approaches to increase computational throughput (Stone et al. 1980).

The area of parallel processing has showed a great deal of promise for such broad applications as military defense, genetic engineering, artificial intelligence, and medical diagnosis. Parallel processing employs two or more elements for the efficient computation of some equation or set of equations through the use of concurrent events. In particular, systolic arrays use two or more individual arithmetic units operated in parallel for highly concurrent processing. Systolic arrays, which are single instruction multiple data architectures, provide a means of optimizing an algorithm for specific implementation in integrated circuits. Systolic algorithms are usually constructed as a



set of identical operations that can be performed in parallel. For this reason, matrix computations are particularly well suited to systolic arrays. More than one element of a resultant matrix can be computed simultaneously to reduce the overall computation time of an application.

The systolic architectural concept was developed by Kung and associates at Carnegie-Mellon University. In a systolic system, data passes from the computer memory through many processing elements (PEs) before it is returned to memory. An analogy is often made between the systolic array and the heart. Data flows through the PEs in a rhythmic fashion similar to the blood circulation in a heart (Briggs and Hwang 1984).

Many special purpose systolic processors have been designed by various universities and industrial organizations. Because they involve several identical building blocks which are used repetitively with simple interfaces, systolic arrays result in cost-effective, high-performance, special-purpose systems for a wide range of potential applications (Briggs and Hwang 1984).

The fundamental principle of a systolic system is quite simple. Replacing a single processing element with an array of processing elements that operate in parallel will result in a higher computational throughput without an increase in memory bandwidth. The memory bandwidth is not increased because exhaustive use of data read from memory is made before the new data is written to memory. In general, the memory only interfaces to the boundary elements of a systolic array. Data flows between



neighboring elements in a pipelined manner. The ability to keep the pipeline full is a measure of the systolic array's efficiency (Kung 1982).

The processing element of a systolic array is essentially an arithmetic logic unit (ALU) with a set of working registers, possibly some scratch pad memory, and a local microcode memory and controller (Briggs and Hwang 1984). In most applications, the PE is scaled down to perform only the set of instructions needed for that particular application. In this paper, the general purpose PE developed by Condorodis will be employed with only modest changes (Condorodis 1987). These changes will be outlined during the design synthesis of this paper.

The PE of Condorodis is capable of high speed calculations and data routing between neighboring elements for the implementation of an orthogonal array. The ALU can perform multiplication, division, addition, subtraction, square, and square root operations at a very high speed on 20-bit logarithmic numbers. The PE also allows the routing of data to neighboring PEs to allow the configuration of various algorithms. It includes a microcode RAM for the local programming of algorithms such as matrix multiplication and inversion.

### The Logarithmic Number System

When a large dynamic range and high precision are required as in most Kalman filtering applications, a floating point number system is usually adopted. Unfortunately, floating point operations are inherently slower than fixed point

operations due to the normalization and denormalization that must be performed before and after arithmetic computations. Taylor developed an ALU based on the Logarithmic Number System (LNS) that is capable of performing multiplication and division faster than conventional floating point ALUs (1985). This ALU forms the basis of the PE designed by Condorodis.

Numbers in LNS are represented with a signed radix raised to some signed exponent. Therefore, multiplication and division operations are simply an addition or subtraction of the exponents, and square and square root operations are simply a left or right shift of the data word. If the radix is constant, a number can be represented in logarithmic notation as a signed exponent alone. For digital computer applications, the radix will be two.

The primary disadvantage of the LNS is that addition and subtraction require the use of memory look-up tables. Thus, the memory required to implement an LNS based ALU with a reasonable dynamic range has been prohibitive. However, the recent advances in VLSI technology and LNS ALU algorithms have made it feasible to consider a single chip implementation of a 20-bit LNS ALU. Addition and subtraction read only memory (ROM) tables can be included on chip when memory reduction techniques are employed (Condorodis 1987).

Although LNS is an ideal numbering system to be used in the implementation of the systolic array Kalman filter, it has become neither an industry nor a military standard. Therefore, to employ the LNS in a Kalman filter machine that must interface



with other systems, it is necessary to convert between LNS and a standard number system. Floating point number systems are commonly used, and the Institute of Electrical and Electronic Engineering (IEEE) has developed a standard for 32-bit floating point numbers. Taylor has shown how to convert between the LNS and floating point formats (1985).

## CHAPTER II, DESIGN FUNDAMENTALS

The Kalman filter equations are natural for a systolic implementation. The initial equations were minimized by Graham and Kadela to reduce the necessary matrix operations (1985). These reduced equations will be used for the Kalman filter implementation.

Matrix inversion has been a stumbling block in previous Kalman filter designs. However, the symmetric nature of the matrix that is inverted in the Kalman equations makes it possible to introduce a simplified, pipelined inverse algorithm based on the Cholesky *LU* decomposition. Furthermore, by using a reconfigurable systolic array, all matrix operations can be performed with the same architecture without the need for purging the systolic pipeline when new instructions are executed.

Control of the individual PEs is one of the most difficult design factors associated with systolic arrays. Although these architectures provide simple network structures for data flow, partitioning, scheduling, and synchronizing of the problem are necessary for correct operation. These factors must be considered when the arithmetic algorithms are designed.



### Kalman Filter Equations

Prior to 1960, work in the area of control and estimation theory modelled and analyzed systems in the frequency domain. In the early 1960's statistical modelling was extended to the time domain using state-space notation (Kalman 1960). This method simplified the mathematical and notational models associated with optimal estimation and was useful in providing a statistical description of system behavior. Time domain models also produced a system description closer to physical reality than any of the previous frequency domain models. Today the Kalman filter is one of the most commonly used state estimators for system control.

A linear discrete dynamic system can be described with

$$x_{k+1} = F_k x_k + G_k w_k \quad (1)$$

and

$$z_k = H_k x_k + v_k. \quad (2)$$

The Kalman filter variables are defined in Table 1. The uncorrelated zero-mean white noise sequences  $w(k)$  and  $v(k)$  have the second order properties

$$Q_k = E[w_k w_k^T] \quad (3)$$

and

$$R_k = E[v_k v_k^T]. \quad (4)$$

An estimate of the state of the system defined by equations (1) and (2) can be obtained given initial estimates of the initial state,  $\hat{x}_0$ , and the state covariance,  $P_0$ , and statistical information of the input noise covariance,  $Q_k$ , and the output noise covariance,  $R_k$ .

TABLE 1  
KALMAN FILTER VARIABLES

Variable	Description
$k$	:= 0,1,2...
$x$	:= nx1 state estimate vector
$F$	:= nxn state transition matrix
$G$	:= nxp system input matrix
$Q$	:= pxp input noise covariance
$H$	:= mxn measurement matrix
$R$	:= mxm output noise covariance
$v$	:= mx1 plant noise vector
$w$	:= px1 system noise vector
$z$	:= mx1 measurement vector

The sequential equations modelling the standard Kalman filter become

$$\hat{x}_{k+1} = F_k \hat{x}_k + F_k K_k [z_k - H_k \hat{x}_k] \quad (5)$$

where the Kalman gain,  $K_k$ , is

$$K_k = P_k H_k^T [H_k P_k H_k^T + R_k]^{-1} \quad (6)$$

and the covariance matrix,  $P_k$  is

$$P_{k+1} = F_k P_k F_k^T - F_k K_k H_k P_k F_k^T + G_k Q_k G_k^T \quad (7)$$

These equations can be computed recursively. Equations (5) through (7) are rewritten as

$$x = F x + F K [z - H x], \quad (8)$$

$$P = F P F^T - F K H P F^T + G Q G^T, \quad (9)$$

and



$$K = P H^T [H P H^T + R]^{-1}. \quad (10)$$

Graham and Kadela have restructured the Kalman equations listed above to reduce the required matrix multiplications by 25 percent over that of the original set of equations (1985). The resulting equations are

$$a = F K, \quad (11)$$

$$x = F x + a [z - H x], \quad (12)$$

$$P = [F - a H] P F^T + G Q G^T, \quad (13)$$

$$b = P H^T, \quad (14)$$

and

$$K = b [R + H b]^{-1}. \quad (15)$$

Note that the original equations required sixteen matrix multiplies, but the new equations have reduced this number to twelve. Storing intermediate results which are used more than once into system memory can produce a significant reduction of necessary computations. Equations (11) through (15) require matrix addition, subtraction, multiplication, inversion, and transposition and are well suited to systolic arrays.

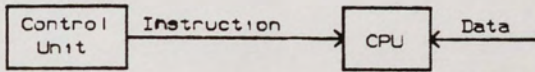
### Systolic Architectures

Parallel computers are frequently classified according to the parallelism within the instruction and data streams. Naturally, there are four types of parallel computers that arise from this classification. The single instruction single data (SISD) machine is a serial computer. In this architecture, only one instruction can execute at any given

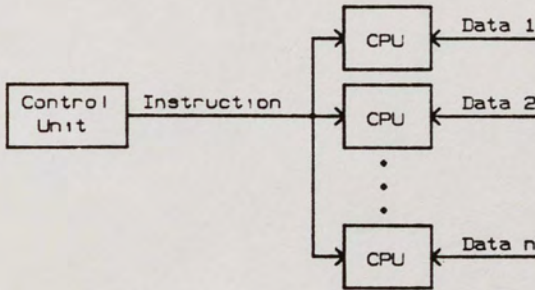
instance. The data stream can be replicated to produce the single instruction multiple data (SIMD) computer. Systolic arrays are of the SIMD architecture. Parallelism in the instruction stream will produce a multiple instruction single data (MISD) processor. In the MISD processor, a data word is operated on by several instructions simultaneously. This type of machine has found limited applications. Parallelism in both the instruction and data streams results in a multiple instruction multiple data (MIMD) computer. The MIMD machine is the most powerful and the most difficult to control. Control of the MIMD device becomes more complex as interconnection between processors increases (Stone et al. 1980). Models of the four classes of parallel computers are shown in Figure 1.

A systolic array consists of many PEs connected to adjacent PEs in a regular fashion. The essential point of the systolic architecture approach is that data is used in each cell it passes for the attainment of some computational goal. Unfortunately, it is not always possible to achieve 100 percent efficiency in PE usage. For example, using a  $n \times n$  systolic array to compute  $m \times m$  matrix addition where  $m < n$  will result in less than maximum PE utilization. However, systolic architectures can often produce dramatic performance improvements even without peak efficiency. Figure 2 contrasts a conventional processor with a systolic array processor. The conventional machine is a Von Neumann machine with one processor while the systolic array processor is a Von Neumann machine with several processors.

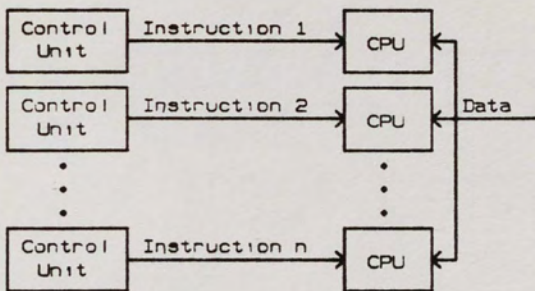




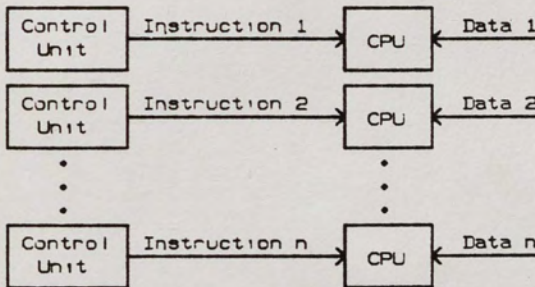
(a) SISD computer mode



(b) SIMD computer model

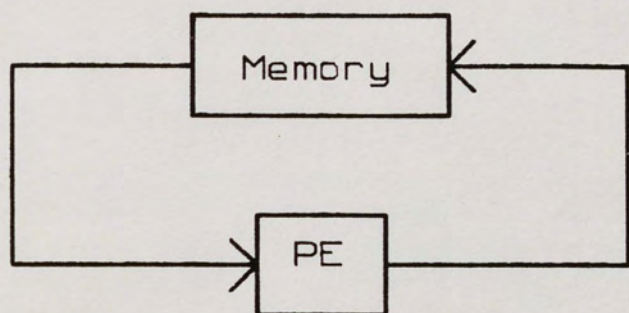


(c) MISD computer model

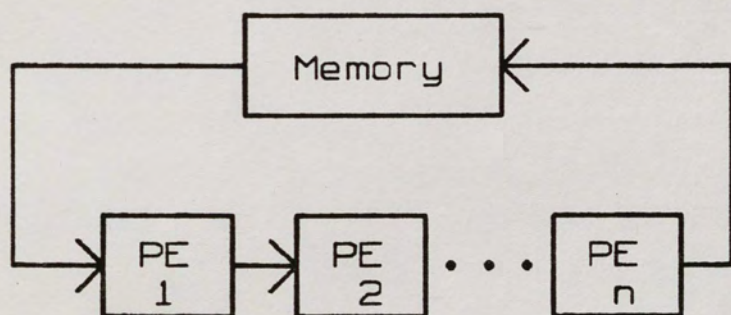


(d) MIMD computer model

Figure 1. Models For Four Computer Classes.



(a) Conventional Processor



(b) Systolic Array Processor

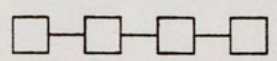
Figure 2. Conventional Versus Systolic Processor Architectures.



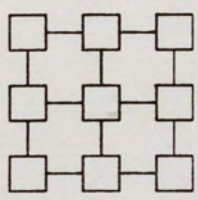
If each PE operates with a clock frequency of 10 MHz, the conventional memory-processor machine can produce at best a performance of five million operations per second (MOPs). A systolic machine operating at the same clock frequency will result in a performance of  $10n/2$  MOPs where  $n$  is the number of PEs used. To take full advantage of the parallel processing feature of systolic arrays, algorithms must be devised as a number of identical computations. In addition to higher performance, systolic arrays offer the advantages of modular expansion, regular data flow, and use of identical processing elements (Briggs and Hwang 1984).

Various geometric configurations exist for systolic architectures that make it possible to implement a variety of algorithms. In addition, a reconfigurability feature can be added to provide on-line operational modifications for the implementation of different algorithms. For example, the systolic array might be used for performing a matrix operation such as  $AB + CD$  where it has to multiply and add matrices. Figure 3 depicts a few of the more common configurations of systolic architectures. Table 2 matches common systolic architectures to some of their applications (Hwang and Briggs 1984). Many problems can take advantage of the performance improvements offered by such systolic topologies by rearranging the flow graphs of those algorithms into recursion relations.

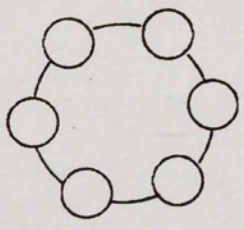
Obviously, if high performance is a primary design objective, then some form of parallel processing must be employed. Traditionally, the SIMD and MIMD structures have been chosen. Recently, systolic arrays have become popular due to many of the



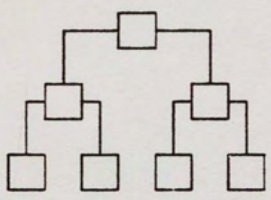
(a) Linear array



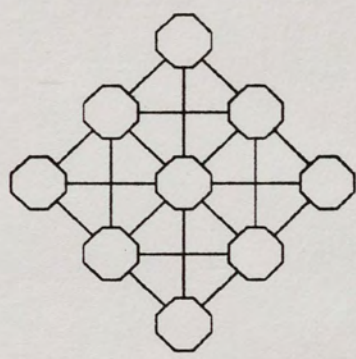
(b) Orthogonal array



(c) Ring



(d) Tree



(e) Octagonal array

Figure 3. Common Systolic Architecture Configurations.



TABLE 2

## GEOMETRIC CONFIGURATIONS AND CORRESPONDING FUNCTIONS

SYSTOLIC ARCHITECTURE	APPLICATION
linear arrays	discrete Fourier transforms, priority queues
orthogonal arrays	orthogonal matrix arithmetic, graph algorithms involving adjacency matrices
hexagonal arrays	band matrix arithmetic, transitive closure, pattern matching, relational database operations
trees	searching algorithms, parallel function evaluation, recurrence evaluation
triangular arrays	inversion of triangular matrix, formal language recognition

reasons described above. The implementation of a basic matrix operation such as a fixed multiply-add function would be well suited for a systolic architecture. However, there are several design obstacles to overcome when considering such an architecture. The lack of much previous experience in the systolic field means that no formal body of knowledge is available detailing the synthesis of such an array. Although systolic arrays of infinite size and ideal qualities can be conceptualized, it is not well understood what happens when the array dimension is reduced to practical limits. In addition, whenever the arithmetic operation of a program changes, it might be necessary to "flush" the systolic array to avoid the interaction of inappropriate data.

The global control of the computational units has also been a problem. In addition, complex algorithm mapping into systolic arrays often yields low PE utilization, especially for orthogonal architectures. All of these problems can be overcome with the introduction of the few instruction multiple data (FIMD) systolic architecture. The FIMD architecture takes advantage of efficient software techniques to perform high level pipelining so many operations can be performed simultaneously yielding a higher PE utilization and throughput. A five by five pipelined, orthogonal systolic array of this type is shown in Figure 4. The reconfigurable, pipelined, orthogonal, systolic array FIMD architecture will be used to present the algorithms of this paper.

#### Arithmetic Algorithms For Systolic Arrays

Arithmetic algorithms are required such that when implemented in orthogonal systolic architectures, they can perform recursive matrix operations at a very high rate. In particular, it is necessary to load a matrix into the structure, to add or subtract two matrices, to multiply two matrices, to invert a matrix, and to transpose a matrix. With the exception of inversion, the pipelined systolic structure should be able to perform all of these operations on non-square as well as square matrices. The  $n \times n$  systolic array should also be capable of processing any of the operations on matrices with dimensions less than the maximum dimension of the array,  $n$ . For Kalman filtering, the symmetric nature of the matrix that is to be inverted can be taken advantage of to simplify the inversion algorithm.



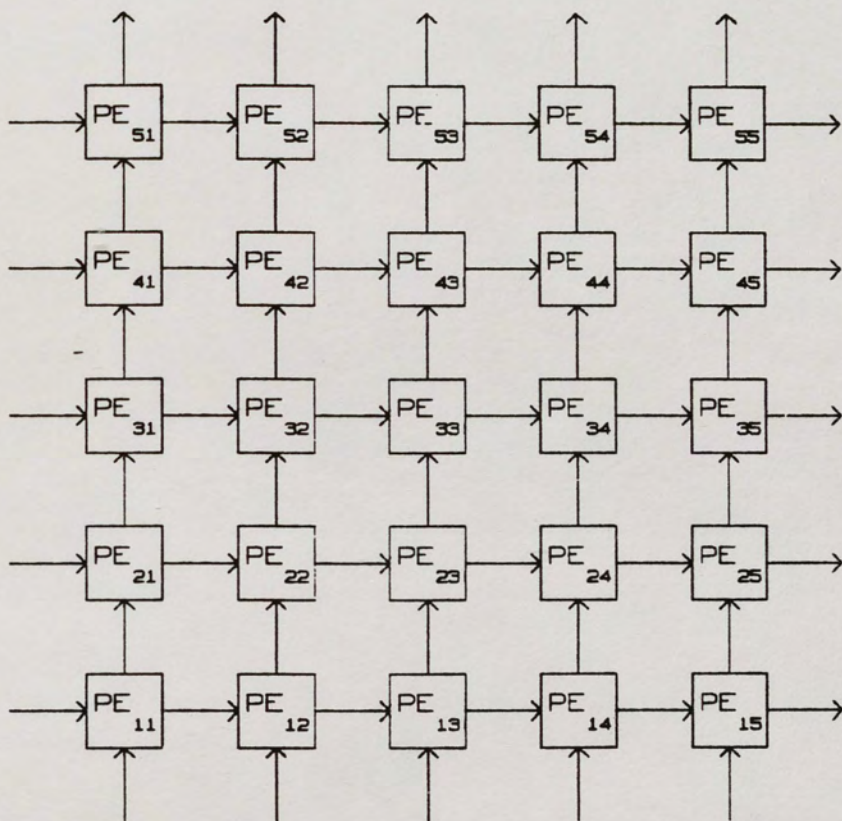


Figure 4. Systolic Pipelined Orthogonal Array.

### Matrix Load

Loading a matrix into the orthogonal array is necessary for performing matrix multiplication, addition, subtraction, and inversion. A matrix is piped into an orthogonal array from the bottom. As a data element passes the appropriate processing element, it is stored into the scratch pad memory of that PE.

The loading order of an array is vital. As shown in Figure 5, the data is loaded into each column with the  $n$ th row loaded first. Columns are loaded in a skewed manner as shown such that element  $a_{52}$  is stored one clock after element  $a_{51}$ ,  $a_{53}$  is stored one clock after  $a_{52}$ , and so on. This is necessary to keep the pipeline full during multiplication and inversion computations as will be discussed later.

With this loading algorithm, element  $a_{51}$  is input to  $PE_{11}$  during the first clock cycle, elements  $a_{41}$  and  $a_{52}$  are input to  $PE_{12}$  and  $PE_{21}$ , respectively, during the second clock, and so on. Therefore, element  $a_{51}$  passes through  $PE_{11}$ ,  $PE_{21}$ ,  $PE_{31}$ , and  $PE_{41}$  before it is stored in element  $PE_{51}$ . This process is repeated for all matrix elements, and once  $PE_{11}$  has stored its matrix data, element  $a_{11}$ , it is ready to begin processing on the next clock cycle.

Note that this loading scheme is systolic since data is passed through the lower elements just to arrive and get stored in the upper elements. Each  $PE_{ij}$  passes  $n - i$  data words before it stores the appropriate matrix data in its local memory. Since a PE passes a variable number of words which depends on the row which it resides, it is necessary to program each row of PEs separately.



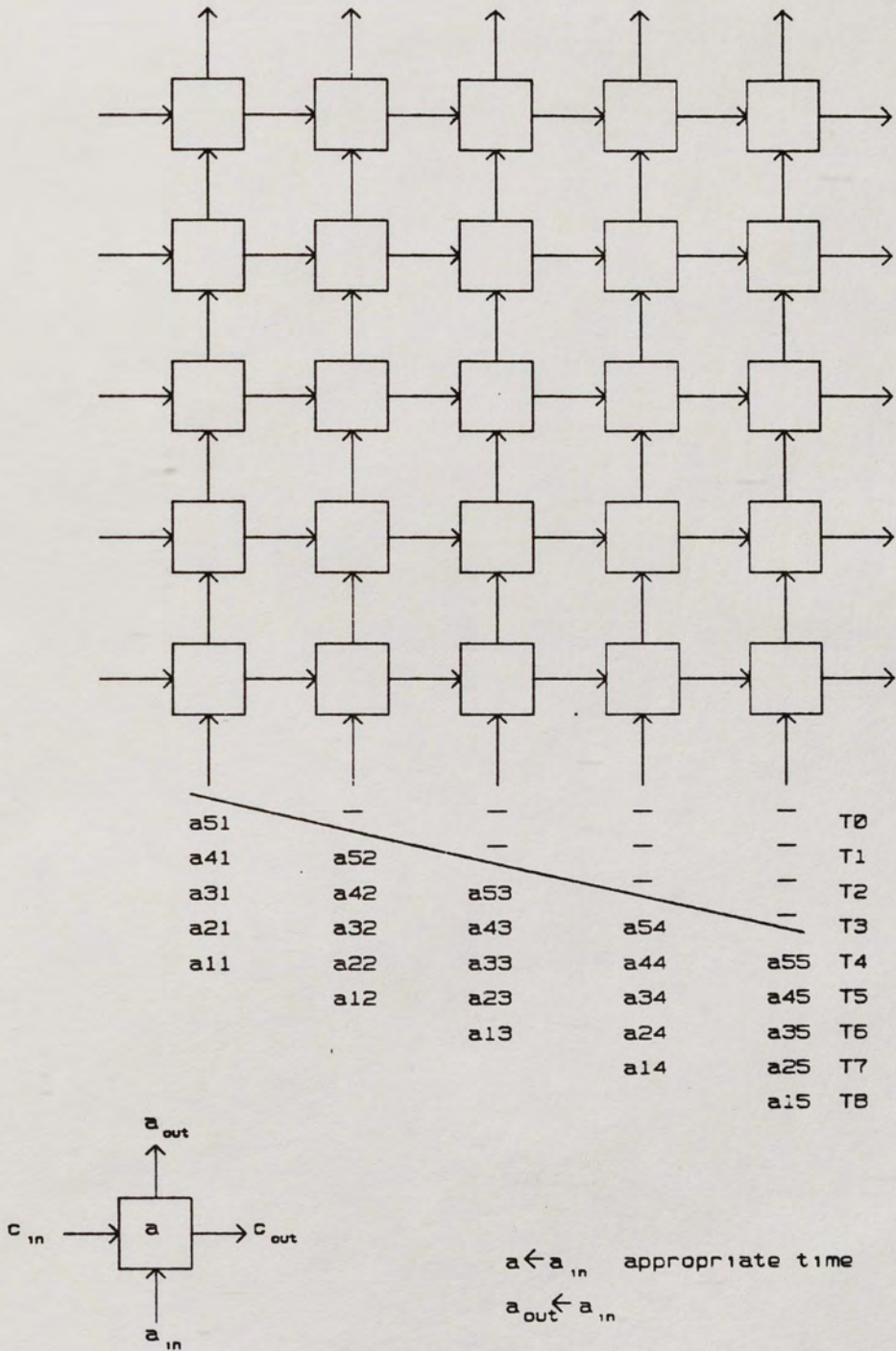


Figure 5. Matrix Load Procedure.

## Matrix Multiplication

The systolic architecture intended for the Kalman filter application must perform matrix arithmetic operations, and many algorithms exist that implement matrix functions using an orthogonal array architecture. One such algorithm developed by Kung obtains the product of two  $n \times n$  matrices in  $4n - 1$  computational units (1982). Recently, a new algorithm was developed by Papadourakis and Taylor that uses the FIMD concept and extensive pipelining. With this procedure, the matrix multiplication throughput is  $2n$  clocks (1986).

Matrix multiplication, addition, and subtraction as presented by Papadourakis and Taylor are identical operations in a pipelined orthogonal systolic architecture. For given matrices  $A$  and  $B$ , an element of the matrix product  $Z$  can be found with the recurrent equation

$$z_{ij} = \sum_{k=1}^n a_{ik} b_{kj}, \quad (16)$$

where  $n$  is the column dimension of  $A$ . Assuming the matrix  $A$  is partially loaded inside the orthogonal array using the procedure described in the previous section,  $B$  can be piped in the system to interact with  $A$  before loading is complete. The first row elements of matrix  $B$  are input from the bottom into the first column of the systolic array with the  $b_{11}$  element input the first clock after the last column one  $A$  element,  $a_{11}$ , is loaded. This pipelining produces a more efficient machine than would otherwise be capable.



During the multiplication operation, each appropriate  $PE_{ij}$  takes the sum of partial products from the left neighbor and adds it to the product of  $a_{ik}b_{kj}$ . Thus, each product matrix element is obtained by accumulating data in the rows of the systolic array from left to right as shown in Figure 6. In addition, the  $B$  data is piped one row deeper into the array. The resulting product matrix is output to the right.

As mentioned, the product of two  $n \times n$  matrices,  $Z = AB$ , has a computation time of  $2n$  clock cycles. This can be verified by the fact that  $n$  units are needed to compute  $z_{11}$  and  $n$  units are required for the partial loading of another matrix.

In general, the product of a  $n_1 \times m_1$  matrix  $A$  and a  $n_2 \times m_2$  matrix  $B$  can be obtained in  $n_1 + m_2$  clocks provided  $m_1 = n_2$  and all matrix dimensions of matrices  $A$  and  $B$  are less than or equal to the largest dimension of the systolic array,  $n$ . Given the same constraints,  $n_1 + m_1$  PEs are utilized during the matrix multiplication.

The operation  $Z = AB + C$  can be performed in  $2n$  clocks since the  $C$  matrix can be input from the left. This requires no more overhead since the left column of the orthogonal array has no left neighbors to input a sum of partial products from. Thus, it is the left column that adds the  $C$  values to the product (Papadourakis and Taylor 1985). This method can be used for attaining the sum or difference of two matrices as well. If an identity matrix is preloaded into the local PE memory, only  $n$  clocks are required for any  $B + C$  or  $B - C$  operation. This operation is identical to the  $AB + C$  operation described above with the  $A$  matrix equal to the identity matrix.

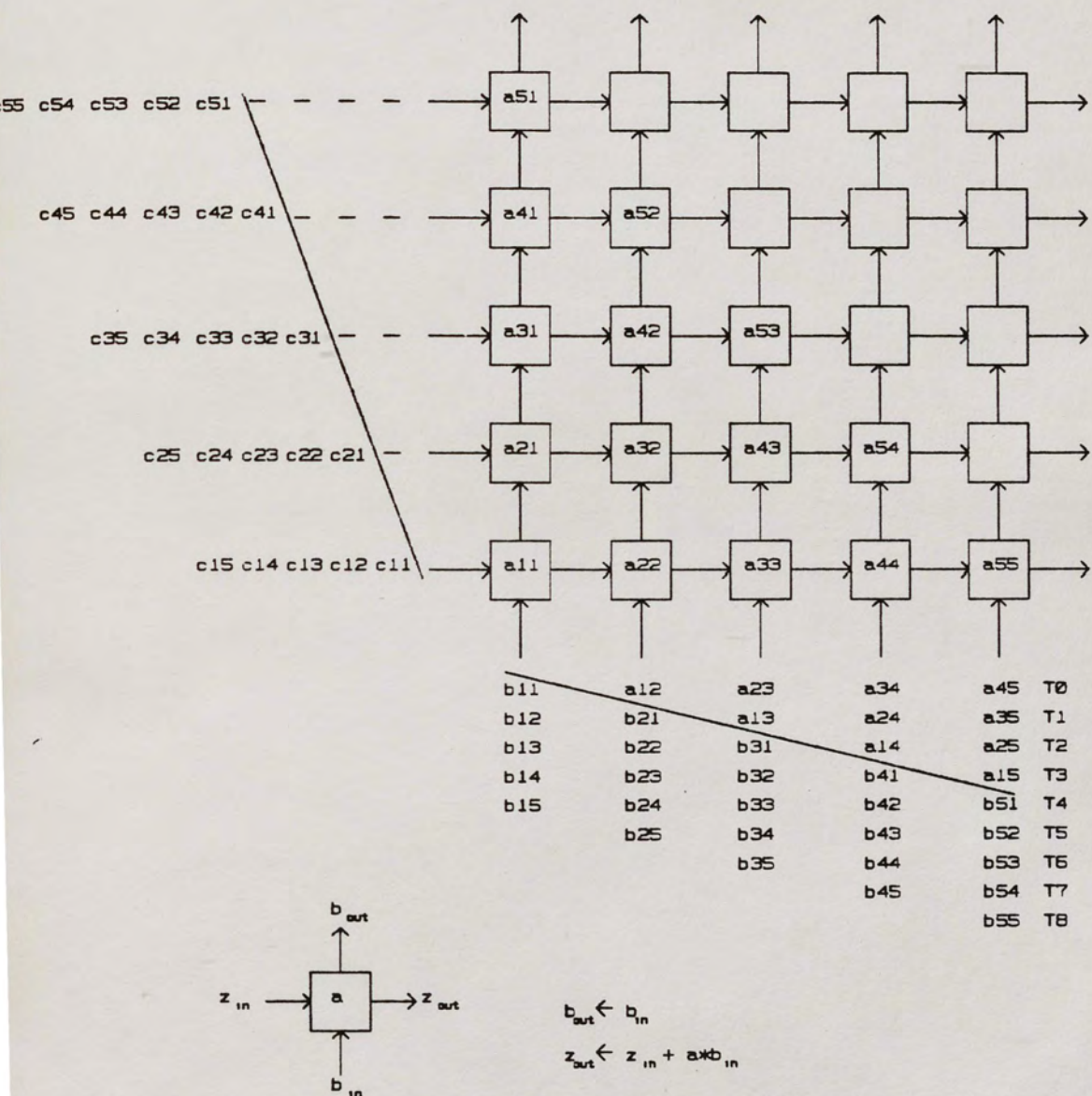


Figure 6. Matrix Multiply Procedure.



### Matrix Inversion

Previous Kalman filter implementations have relied on an iterative inversion algorithm. This algorithm required four iterations or  $16n$  clocks to achieve the desired accuracy for most Kalman filter applications (Graham and Kadela 1985). Using the FIMD architecture, Liu and Young demonstrated a new method for inverting a covariant matrix in an orthogonal systolic array that is based on Crout's triangularization method (1984). This procedure, as demonstrated, is simplified for symmetric matrices when Cholesky's decomposition method is used. Although Cholesky's method requires fewer computations, it has not been very popular because of the overwhelming computational burdens imposed by the need to compute the square root. However, using the proposed LNS computational unit will eliminate this burden, and a very high speed inversion implementation can be developed which will further increase throughput. The matrix requiring inversion in the Kalman filter equations is symmetric. Therefore, it is amenable to this implementation.

A non-singular matrix  $C$  can be decomposed as a product of a lower matrix  $L$  and an upper matrix  $U$ . Then, the inverse

$$C^{-1} = U^{-1} L^{-1}. \quad (17)$$

If the matrix is symmetric and Cholesky's decomposition method is applied, then

$$L = U^T. \quad (18)$$

so

$$C^{-1} = U^{-1} (U^{-1})^T. \quad (19)$$

Triangular matrix inversion can be used to find the inverse of  $U$ .

### Procedure For Upper Matrix Decomposition

The decomposition of matrix  $C$  requires that only the upper triangular portion of the matrix be input since it is symmetric. Cholesky's method can be represented as

$$u_{ij} = \sqrt{c_{ii} - \sum_{k=1}^{i-1} u_{ki}^2} \quad \text{for } i = j \quad (20)$$

and

$$u_{ij} = \frac{c_{ij} - \sum_{k=1}^{i-1} u_{ki} u_{kj}}{u_{ii}} \quad \text{for } i \neq j.$$

The orthogonal array representation of this algorithm is shown in Figure 7. The orthogonal array functionally contains two types of PEs, diagonal and non-diagonal. The diagonal PEs are depicted by circles in Figure 7. These elements calculate and store the square root of the left input and pass the  $u_{ii}$  value to the top output when the first data is received. The diagonal PEs pass  $c_{in} / u_{ii}$  to the PEs directly above it for the remaining cycles, where  $c_{in}$  refers to the data input from the left. The non-diagonal PEs store the incoming  $u_{ij}$  value at the appropriate PE cycle. The following cycles are used to calculate  $c_{in} - uu_{in}$ . The results are passed to the right where they are used in the diagonal PEs to calculate the  $u_{ij}$  values. The  $u_{ij}$  values are piped vertically through the array so that they can be stored in the appropriate PEs and retransmitted to the bottom of the array for the calculation of the inverse of the upper triangular matrix.



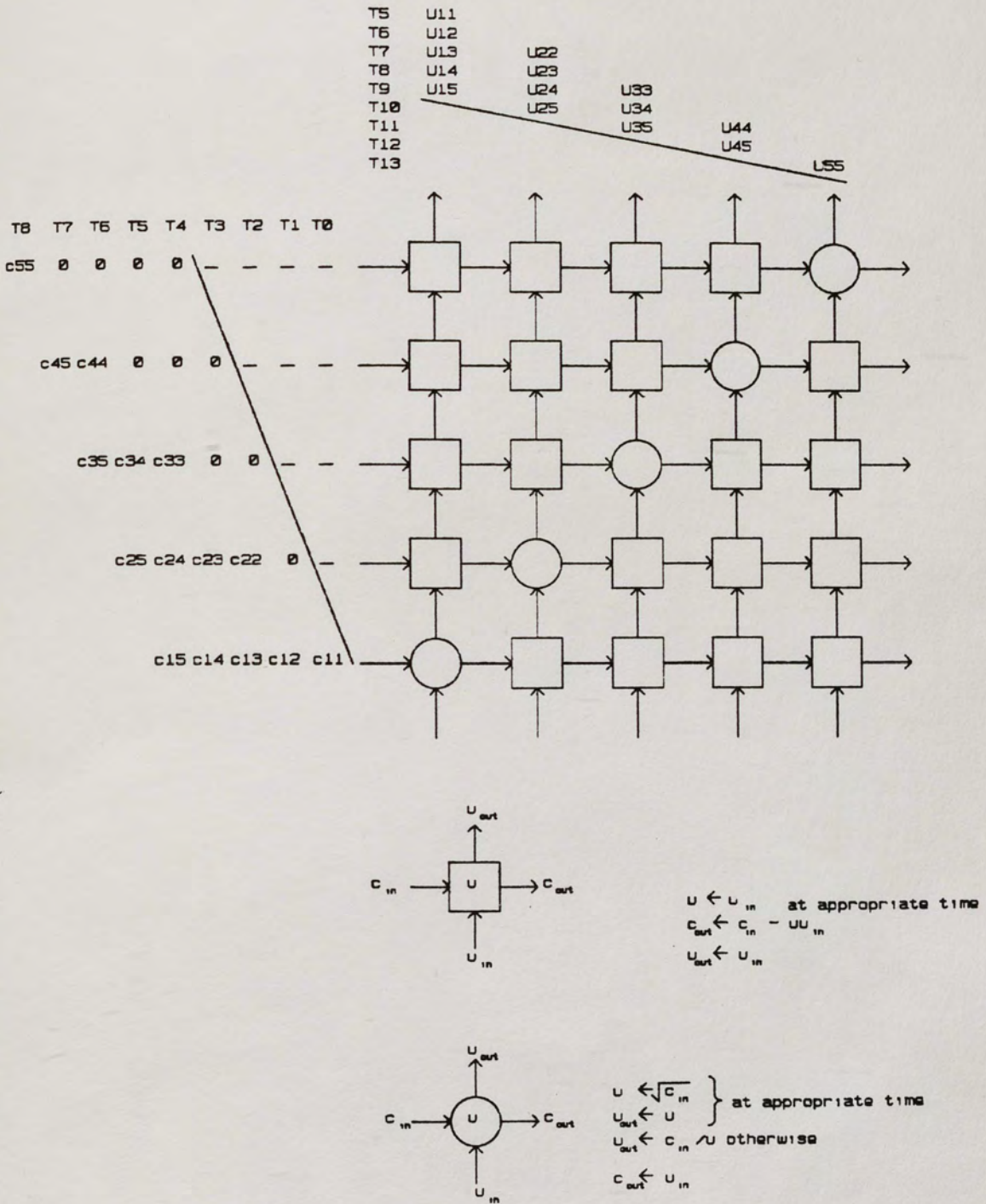


Figure 7. Upper Triangular Matrix Decomposition Procedure.

The upper triangularization time is  $n$  clocks.

### Procedure For Inversion Of Upper Matrix

Exactly  $n$  clock cycles after the start of the upper decomposition, the inverse calculation of the triangular matrix  $U$  can commence. The triangular inverse algorithm can be represented as

$$v_{ij} = \frac{1}{u_{ii}} \quad \text{for } i = j \quad (21)$$

and

$$v_{ij} = -v_{jj} \sum_{k=1}^{j-1} v_{ik} u_{kj} \quad \text{for } i \neq j.$$

The orthogonal array representation of the upper triangular matrix inversion is shown in Figure 8. The PEs calculate and store the ratio  $c_{in}/u_{in}$  during the first PE cycle. The next cycles are used to calculate  $c_{in} - v_{ij}u_{in}$ . The resulting inverse is output to the right. Thus, the  $v_{ij}$  values are stored in the lower triangular PEs of the array. The total upper triangular inverse computation time is  $n$  clocks.

The remaining step is to obtain the product of  $V$  and its transpose. Note that the matrix  $V$  is stored in the appropriate PEs during the calculation of  $V$ . Therefore, reloading is not necessary and only  $V^T$  needs to be input. The procedure for inputting a matrix in a transposed order will be discussed in a later section. A total of  $n$  clocks are needed to complete the multiplication step. Therefore, the total inversion computation time is  $3n$  clocks. The entire inversion procedure is depicted in Figure 9.



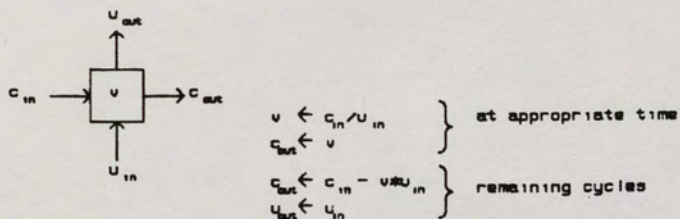
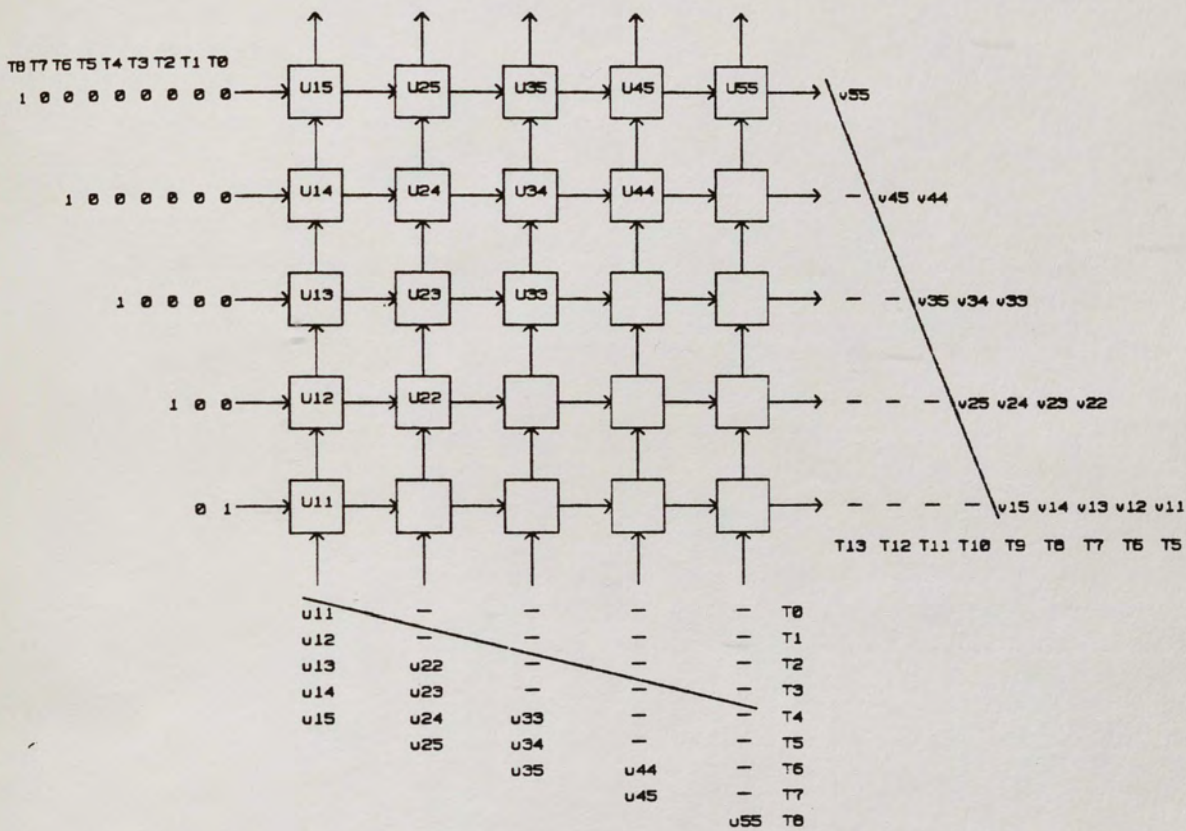


Figure 8. Upper Triangular Matrix Inversion Procedure.

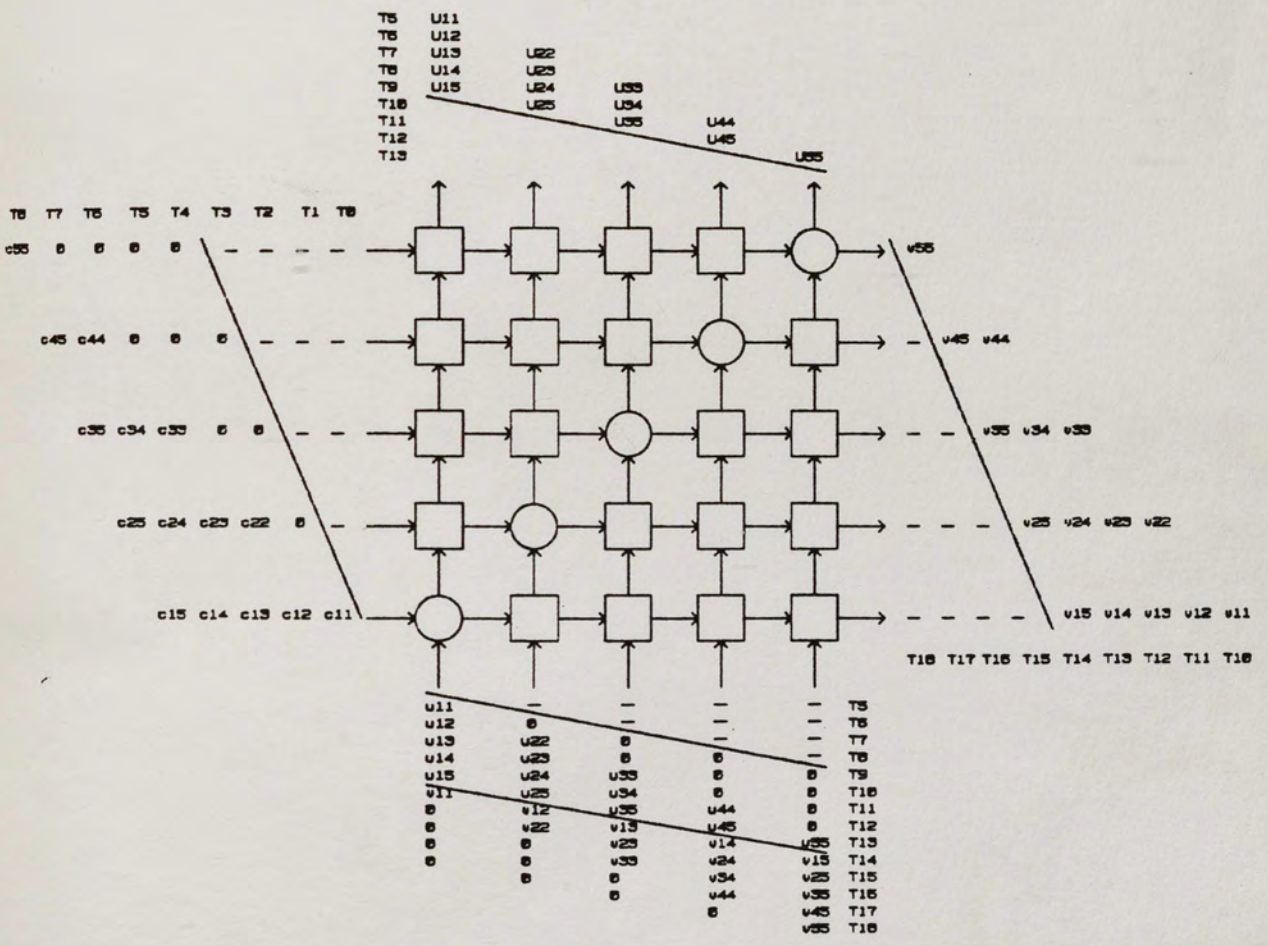


Figure 9. Pipelined Inversion Procedure.



It should be noted that the inversion of a non-singular, non-symmetric matrix can be performed in  $4n$  clocks. The non-symmetric case requires the calculation of the lower triangular matrix and its inverse.

### Example of Inversion Procedure

Often the general equations used to represent an algorithm are difficult to understand. This general  $4 \times 4$  matrix example should serve as an aid in understanding the matrix inversion procedure from a mathematical standpoint.

Consider a matrix

$$R = \begin{bmatrix} r_{11} & r_{12} & r_{13} & r_{14} \\ r_{21} & r_{22} & r_{23} & r_{24} \\ r_{31} & r_{32} & r_{33} & r_{34} \\ r_{41} & r_{42} & r_{43} & r_{44} \end{bmatrix}. \quad (22)$$

Let  $R$  be symmetric as in the Kalman filter case so that  $r_{ij} = r_{ji}$ . The diagonal elements of the upper triangular matrix resulting from decomposition can be found from equation (20) to be

$$\begin{aligned} u_{11} &= \sqrt{r_{11}}, \\ u_{22} &= \sqrt{r_{22} - u_{12}^2}, \\ u_{33} &= \sqrt{r_{33} - u_{13}^2 - u_{23}^2}, \end{aligned} \quad (23)$$

and

$$u_{44} = \sqrt{r_{44} - u_{14}^2 - u_{24}^2 - u_{34}^2}.$$

Similarly, the non-diagonal elements are

$$u_{12} = \frac{r_{12}}{u_{11}},$$

$$u_{13} = \frac{r_{13}}{u_{11}},$$

$$u_{14} = \frac{r_{14}}{u_{11}},$$

(24)

$$u_{23} = \frac{r_{23} - u_{12}u_{13}}{u_{22}},$$

$$u_{24} = \frac{r_{24} - u_{12}u_{14}}{u_{22}},$$

and

$$u_{34} = \frac{r_{34} - u_{13}u_{14} - u_{23}u_{24}}{u_{33}}.$$

The inverse,  $V$ , of the upper triangular matrix can be computed from equation (21).

The resulting diagonal matrix elements are

$$v_{11} = \frac{1}{u_{11}},$$

$$v_{22} = \frac{1}{u_{22}},$$

(25)

$$v_{33} = \frac{1}{u_{33}},$$

and

$$v_{44} = \frac{1}{u_{44}}.$$

The non-diagonal elements are

$$v_{12} = -v_{11}u_{12}v_{22},$$

$$v_{13} = -[v_{11}u_{13} + u_{12}v_{23}]v_{33},$$



$$v_{14} = - [v_{11}u_{14} + v_{12}u_{24} + v_{13}u_{34}]v_{44}, \quad (26)$$

$$v_{23} = - v_{22}u_{23}v_{33},$$

$$v_{24} = - [v_{22}u_{24} + v_{23}u_{34}]v_{44},$$

and

$$v_{34} = - v_{33}u_{34}v_{44},$$

The required inverse of matrix  $R$  is

$$R^{-1} = V V^T. \quad (27)$$

### Matrix Transposition

Equations (11) through (15) contain four matrix transpositions including the one required for matrix inversion. None of these transposed matrices act as a loading matrix. However, the loading of a transposed matrix will be considered for completeness.

Loading of a transposed matrix may be necessary for such operations as  $A^T B$ . If it is assumed that data elements are input in the pipelined order shown in Figure 5, the simplest way to load the matrix transposed is to input the data from the left of the systolic array in a manner similar to the loading procedure. Thus, element  $a_{11}$  would be stored in  $PE_{11}$ , element  $a_{21}$  would be stored in  $PE_{12}$ , and so on. This is not necessary if the memory controller reads data in a transposed order. However, it is not feasible to have a  $2n$  port memory for  $n > 2$  since this would result in intensive decode circuitry. It will probably be necessary to have separate memory banks for rows or columns of data. The consideration of the external memory and associated controller

is beyond the scope of this paper. However, it will be assumed that  $n$  memory banks will be used to store matrix data.

For an operation such as  $AB^T$ , matrix  $A$  is loaded in the manner described by Figure 5. However, the  $B$  matrix must be input in a transposed manner. As before, if the memory controller can read matrix elements in a transposed order, this method is preferred. Without this memory capability, a transpose switch is necessary to swap  $b_{ij}$  and  $b_{ji}$  as they are input to the array. Note from Figure 6 that  $b_{ij}$  and  $b_{ji}$  are input during the same clock interval for all  $i$  and  $j$ . This makes the implementation of a transpose switch a fairly straightforward procedure. A diagram of a possible implementation of the switch is shown in Figure 10 for a  $5 \times 5$  systolic array.

Upon receipt of the first element of a  $n \times n$  matrix to be input transposed, the counter of the transpose switch is reset and a transpose signal is sent for  $n$  clocks. During the first clock, element  $b_{11}$  is input as usual. During the second clock, the counter is incremented, and the column one 5-to-1 multiplexer selects the column two data, and the column two 5-to-1 multiplexer selects the column one data. This process propagates for  $n$  clocks until the entire matrix is loaded.

### PE Control

When a task is presented to a parallel processor, it must be separated into subtasks that can be processed in parallel. This is the partitioning problem associated with systolic array control. This problem was solved in the previous sections when



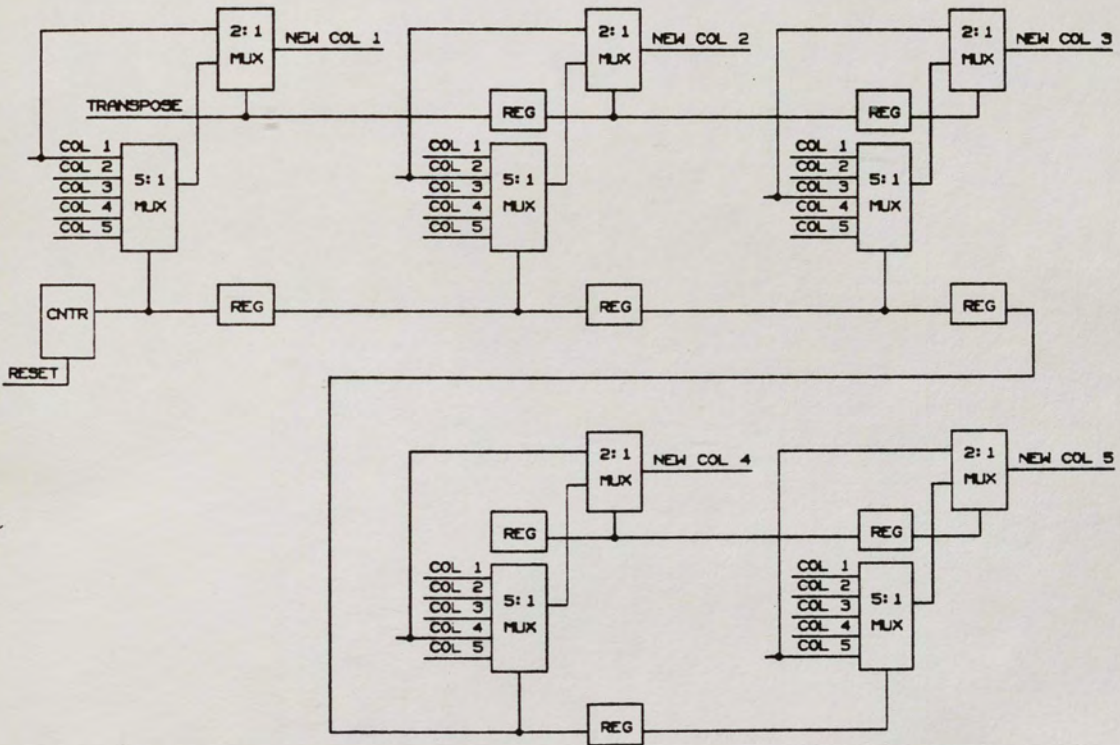


Figure 10. Block Diagram of Transpose Switch.

arithmetic operations in the orthogonal array were described. Efficient partitioning extracts the maximum amount of parallelism from a problem with minimum system overhead. Often tradeoffs exist between these two goals. To achieve maximum parallelism, it might be necessary to divide a task in such a way that requires more setup time than a less than perfect parallel scheme. For example, any algorithm that causes a break in the systolic pipeline would produce inefficiency resulting from processor idle time.

Once partitioned, it is necessary to divide subtasks among the various PEs. This is the scheduling problem and was accounted for in the previous sections. Finally, the individual PEs need to be synchronized with respect to one another. This control issue is often the most complex. The primary goal of synchronization is to have associated data and instructions at the input to a PE during the same clock interval. The synchronization scheme presented here is a data driven wavefront. The PE opcodes are propagated in a wave-like manner as shown in Figure 11. Opcodes are used to tell the individual PEs what function to perform. The general control structure shown in Figure 11 is used for all of the matrix operations. The first opcode corresponding to a particular operation is input to  $PE_{11}$ . The opcode is registered there, used by that PE, and output to  $PE_{12}$  and  $PE_{21}$  during the next clock. Control propagation proceeds through the rest of the array in  $2n - 1$  clocks. Each PE receives  $n$  identical opcodes for each matrix operation. The numbers inside each PE of Figure 11 represent the clock delay associated with the receipt of the first opcode by that PE.



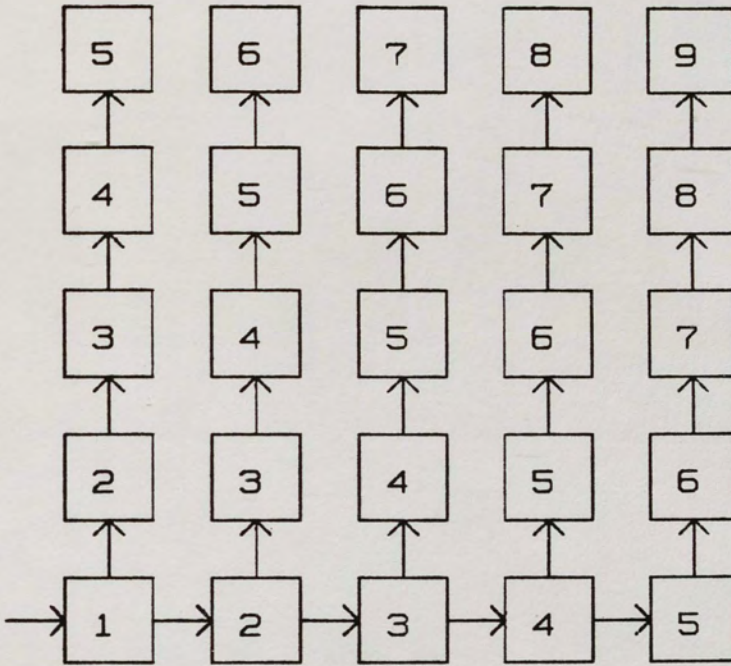


Figure 11. Control Structures For Orthogonal Array Algorithms.

Note that the inversion and loading procedures require that each row of PEs be micro-programmed uniquely. Further, the upper triangularization method requires the diagonal elements to be programmed differently than the non-diagonal elements. Micro-programming each PE uniquely destroys the uniformity of the systolic array and makes the writing of microcode a function of  $n^2$ . However, it does provide significant performance improvements, and the programming burden can be administered to a compiler.

### Processing Element

The processing element is the principle component of the systolic array. The PE must be functionally capable of implementing the algorithms necessary for the Kalman filter application. In addition, it must support the wavefront control outlined in the previous section. The PE of Condorodis meets most of the specifications for the Kalman filter implementation. The general PE to be described below is a modified version of the Condorodis PE. Differences will be noted.

The required PE will employ the LNS to provide high speed arithmetic capability and a large dynamic range. The arithmetic capability will include addition, subtraction, multiplication, division, square, and square root of LNS numbers. In addition, the PE will be capable of routing internal and external data to adjacent PEs. An on chip writable control memory and controller will provide programmability. A block diagram of the PE to be used in the Kalman filter implementation is shown in



Figure 12.

### The LNS ALU

The block diagram of the LNS ALU developed by Condorodis is depicted in Figure 13. Note that it is possible to perform operations such as  $AB + C$  and  $A + B^2$  in one clock since the propagation delay through the adder-multiplier and adder-square paths is less than one clock period.

### LNS Arithmetic

Numbers in LNS are represented as

$$X = (-1)^{S_{rx}} r^{e_x}, \quad (28)$$

where  $r = 2$ . In equation (28),  $e_x$  is a 19-bit two's complement number with a 6-bit integer part and a 12-bit fractional part. The radix sign bit is  $S_{rx}$ . LNS numbers are represented in the word format shown below.

$S_{rx}$	19-bit exponent magnitude (two's complement)
----------	---

To represent  $x = 0$  logarithmically, it is necessary to provide a zero flag with each data that is set for this special case (since  $\log(0) = \infty$ ).

The product of logarithmic numbers  $X$  and  $Y$  may be written as

$$Z = X Y = (-1)^{S_{rx} + S_{ry}} r^{e_x + e_y} \quad (29)$$

where

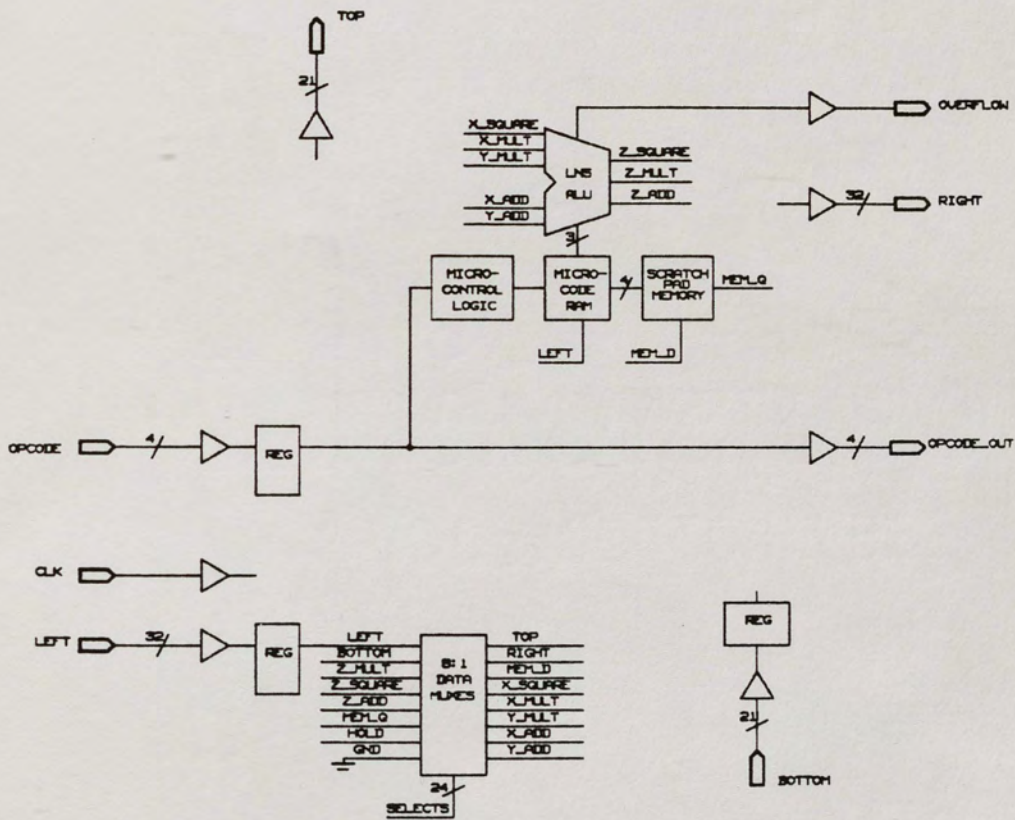


Figure 12. Block Diagram of Processing Element.



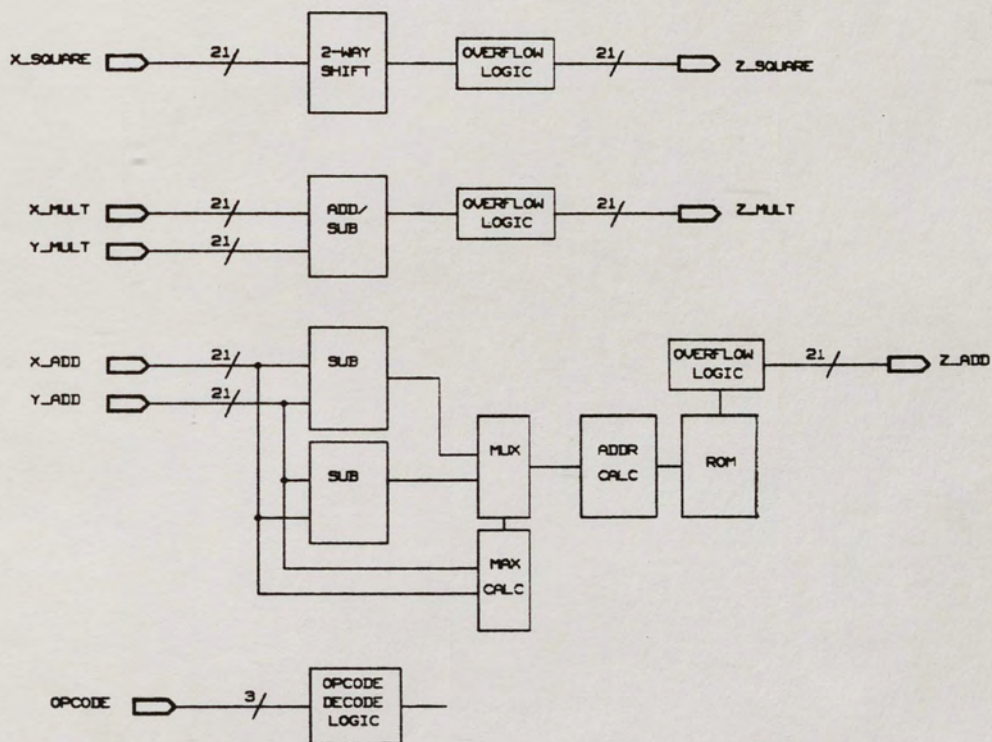


Figure 13. Block Diagram of LNS ALU.

$$S_{rz} = S_{rx} + S_{ry} \text{ and } e_z = e_x + e_y. \quad (30)$$

Thus, the product of two logarithmic numbers simply requires the addition of the exponent values of the two numbers.

Division is similar to multiplication in the LNS. The quotient of two LNS numbers is

$$Z = X / Y = (-1)^{S_{rx} - S_{ry}} r^{e_x - e_y} \quad (31)$$

where

$$S_{rz} = S_{rx} - S_{ry} \text{ and } e_z = e_x - e_y. \quad (32)$$

Thus, the quotient of two logarithmic numbers is obtained by subtracting the exponent values of the numbers.

The addition of two logarithmic numbers is not as straightforward as either multiplication or division. The sum of two LNS values can be expressed as

$$Z = X + Y = (-1)^{S_{rx}} r^{e_x} + (-1)^{S_{ry}} r^{e_y}, \quad (33)$$

or

$$Z = X + Y = (-1)^{S_{rx}} r^{e_x} (1 + (-1)^{S_{ry} - S_{rx}} r^{e_y - e_x}). \quad (34)$$

Taking the logarithm of both sides of equation (34) produces

$$e_z = e_x + \text{LOG}_2(1 + r^{e_y - e_x}). \quad (35)$$

However, to minimize memory requirements,  $e_z$  is implemented as

$$e_z = e_{\max} + \text{LOG}_2(1 + r^{e_{\min} - e_{\max}}), \quad (36)$$

where  $e_{\max} = \max(e_x, e_y)$  and  $e_{\min} = \min(e_x, e_y)$  (Condorodis 1987). The addition operation requires the table look-up of the argument of equation (36).



The logarithmic subtraction operation is similar to the addition operation as it also requires a table look-up in memory. The difference of two numbers  $X$  and  $Y$  can be expressed as

$$Z = X - Y = (-1)^{S_x} r^{e_x} - (-1)^{S_y} r^{e_y}, \quad (37)$$

or

$$Z = X - Y = (-1)^{S_x} r^{e_x} (1 - (-1)^{S_y - S_x} r^{e_y - e_x}) \quad (38)$$

The resulting difference can be represented as

$$e_z = e_x + \text{LOG}_2(1 - r^{e_y - e_x}). \quad (39)$$

Equation (39) is similar to equation (35) and can be restructured as

$$e_z = e_{\max} + \text{LOG}_2(1 - r^{e_{\min} - e_{\max}}) \quad (40)$$

to reduce the memory requirements. Letting

$$D = e_{\max} - e_{\min} \quad (41)$$

allows the equations for addition and subtraction to be represented as  $\text{LOG}_2(1 + 2^D)$  and  $\text{LOG}_2(1 - 2^D)$ , respectively. These two functions can be implemented in the PE by using a table look-up ROM with  $D$  as the address input. The ROM will consist of separate tables for the addition and subtraction functions shown above. Condorodis employed memory reduction techniques on the two functions to take advantage of approximately linear regions. This technique reduced the total memory requirements for the table look-up procedure from 9.5 Mbits to 154 Kbits. The memory requirements for the addition function are shown in Table 3 (Condorodis 1987). The memory requirements for the subtraction function are identical

TABLE 3  
MEMORY REQUIREMENTS FOR LOGARITHMIC ADDITION

ROM ID	D RANGE	ADDRESS RANGE	ROM SIZE	TOTAL BITS
1	0.0-0.5	0-2047	2K x 11	22 Kbits
2	0.5-1.0	2048-3071	1K x 11	11 Kbits
3	1.0-2.0	3072-5119	2K x 12	24 Kbits
4	2.0-3.0	5120-6143	1K x 11	11 Kbits
5	3.0-4.0	6144-6655	512 x 10	5 Kbits
6	4.0-5.0	6656-6911	256 x 9	2.3 Kbits
7	5.0-6.0	6912-7039	128 x 8	1 Kbits
8	6.0-7.0	7040-7103	64 x 7	.45 Kbits
9	7.0-8.0	7104-7135	32 x 6	.2 Kbits
10	8.0-9.0	7136-7151	16 x 5	.08 Kbits

The square and square root operations are necessary for the matrix inversion algorithm. In LNS the square function is simply a single bit left shift since  $LOG_2(x^2) = 2 LOG_2(x)$ . It may be expressed as

$$e_z = 2^{e_x} \ll 1 \text{ bit.} \quad (42)$$

Similarly, the square root function is just a single bit right shift, and it may be expressed as

$$e_z = 2^{e_x} \gg 1 \text{ bit.} \quad (43)$$



### ALU Functional Description

The LNS ALU performs the six matrix operations described above on 20-bit LNS words. The multiplication and division operations are performed with an adder and a subtractor, respectively. The addition and subtraction functions are calculated with an on chip ROM and a special addressing scheme that performs a table look-up. The square and square root operations are performed with a zero fill left shifter and a sign extended right shifter, respectively. The ALU can accept up to five data inputs, one for the square/square root block, and two each for the multiplier/divider and adder/subtractor sections. Included with each data input is a zero flag. In addition, the ALU inputs a 3-bit instruction opcode. The ALU instruction opcode definition is shown in Table 4.

TABLE 4  
OPCODE DEFINITION FOR ALU

OPCODE	OPERATION
0xx	divide
x0x	square root
xx0	subtract
1xx	multiply
x1x	square
xx1	add

The output of the ALU is a 20-bit LNS number, a zero flag, and a system overflow flag. Overflow logic is used to handle the special case when the result of some arithmetic operation is larger than the system was designed to support. Under such a condition, the ALU outputs the largest representable number and sets the overflow flag to a logic one.

Eight-to-one multiplexers are present at the inputs of the various ALU data paths. Each multiplexer is controlled with a 3-bit opcode which selects the source of data. The opcode definition of these multiplexers is shown in Table 5. Note one exception to Table 5. The multiplexer for the top output selects the bottom input when a zero opcode is received. This default allows data to be piped from bottom to top during no operation conditions. The left input is selected when the opcode is a one. This multiplexing scheme is different from that proposed by Condorodis. The previous PE

TABLE 5

## OPCODE DEFINITION FOR INTERNAL DATA MULTIPLEXERS

OPCODE	SOURCE
000	left input
001	bottom input
010	multiply/divide output
011	square/square root output
100	add/subtract output
101	scratch pad output
110	previous data input (hold)
111	ground



design used bus switches to control data input to the ALU, scratch pad memory, and output ports. The old scheme would be difficult to implement since it would require eleven internal tristate busses. Generally, the tristating of data lines is slow. The new scheme makes microcode programming of the PE a simpler task since the data input to the ALU, scratch pad memory, and output ports is controlled by identical multiplexers. Microcode memory requirements are reduced since the new microcode width is 32 bits rather than 46 bits.

#### Scratch Pad Memory

In conjunction with the ALU, a scratch pad memory is provided to store internal data for more than one clock. This data can be an element of a loading matrix, an intermediate value of some operation, or a constant such as zero or one. The scratch pad memory is 8-words by 21-bits. Included with each word is the zero flag of the data. The scratch pad memory is controlled by a 3-bit microcode address and a microcode write enable signal. Data written to the memory is selected with an 8-to-1 multiplexer which is identical to that described in the previous section.

#### PE Control Structure

The internal control structure is a very important feature of the PE. It determines the functional power of the PE as well as its relative ease to program. It also helps to simplify the external control. The PEs make use of a microcontroller and a microcode RAM for government of the functions of the PE.

### The PE Microcontroller

The PE includes a 1K-word by 31-bit microcode RAM to allow the programming of the PE for implementation in a systolic array. A microcode controller provides the capability to partition the microcode memory into four groups for the processing of different functions. The ability to partition the memory into control groups means that all PEs of a systolic array can be loaded with the same microcode, but various PEs can access different control groups within the memory. This feature is not used in the Kalman filter implementation.

A start and end address associated with each of the four groups must be input to the PE. A counter is used to increment the address of the microcode RAM as the PE executes instructions. Comparators are used to keep the address between the start and end addresses. When the counter reaches the end address, it simply wraps back to the start address. Thus, recursive operations can be repeated indefinitely.

The instruction opcodes for the external PE control are defined in Table 6. The loading of the microcode memory can be accomplished by inputting the start and end address for counter one. Microwords from the left input port are written consecutively to locations within the range specified by the end and start address when the opcode is fifteen. The counter is incremented after each write.



TABLE 6  
OPCODE DEFINITION FOR PE CONTROL

OPCODE	FUNCTION
0000	no operation
0001	run counter 1 microcode
0010	run counter 2 microcode
0011	run counter 3 microcode
0100	run counter 4 microcode
0101	load counter 1 start address
0110	load counter 2 start address
0111	load counter 3 start address
1000	load counter 4 start address
1001	load counter 1 end address
1010	load counter 2 end address
1011	load counter 3 end address
1100	load counter 4 end address
1101	pass data to right
1110	read from memory
1111	write to memory

### The Control Word

The PE control word can be subdivided into an upper control word and a lower control word. Each control word is 16 bits. The upper control word is shown below.

31	30	28	27	25	24	22	21	19	18	16
UNUSED	ALU	MULT_X	MULT_Y	SQUARE_X	ADD_X					

The bit range of each field is shown above it. The ALU field controls the ALU operations according to the definition of Table 4. The remaining fields act as a 3-bit select to the respective 8-to-1 multiplexer.

The lower control word is shown below.

15	13	12	10	9	7	6	4	3	2	0
ADD_Y	TOP	RIGHT	MEM	W/R	MEM_ADDR					

The W/R signal is the write enable to the scratch pad memory. If it is set, data will be written to the location specified by MEM\_ADDR. This address field is also used for reading data from the single port scratch pad memory. The remaining fields act as 3-bit selects to the respective 8-to-1 multiplexer.



## CHAPTER III, DESIGN ANALYSIS

With the fundamental Kalman filter and systolic concepts well understood, it is now possible to examine the Kalman filter implementation in more detail. The architecture presented will be reconfigurable so that a minimum of hardware will perform all of the necessary operations and no "flushing" will be necessary to switch operational modes. In addition, the architecture will provide FIMD capability so the total power of parallel architectures will be realized. The performance and efficiency of the implementation will be measured to determine the success of this project.

### Control of Successive Operations

The solution to the Kalman filter equations involves several steps. In order to perform these steps in succession with maximum efficiency, it is necessary to keep the pipeline as full as possible. There are essentially two types of operations needed to implement the Kalman filter, multiply and inverse operations. All procedures are identical in terms of control flow. The integration of multiply-multiply, multiply-inverse, inverse-multiply, and inverse-inverse steps must be examined.

Successive multiplication operations can be achieved easily in the pipelined systolic architecture. If two matrix products such as  $AB$  and  $AC$  are desired, matrix  $A$  can be loaded and matrices  $B$  and  $C$  can be input in succession. The first product will require  $2n$  clocks as noted previously. However, the second product will only require

an additional  $n$  clocks to obtain since reloading of  $A$  is not required.

Two unique products such as  $AB$  and  $CD$  can be obtained by loading  $A$ , inputting  $B$ , loading  $C$ , and inputting  $D$  in succession. The total computation time is  $4n$  clocks.

All inverse operations require a matrix multiplication as a last step. Consequently, the integration of inverse and multiplication procedures can be viewed as successive multiplication steps.

#### Order of Pipelined Operations

The inverse, multiply, load, and transpose algorithms developed in the previous chapter can be combined in an orthogonal implementation which uses the LNS based PE. The order which the operations of equations (11) through (15) are performed is important. It is necessary to have intermediate values calculated and available when they are needed. For example, equation (14) should be calculated before equation (15) since the latter requires the result of the former. It would be desirable to have the data available at the output of the systolic array rather than in an internal node of the systolic array when it is time to input it for some other computation. The use of switches to route data from internal nodes is costly in terms of hardware and disrupts the data flow within the systolic pipeline.

The recommended order of the Kalman operations is depicted in Table 7. The table shows the size of the matrices involved and breaks the inverse operation into subtasks. The start time of each operation is represented in terms of the general



TABLE 7  
KALMAN FILTER OPERATIONS VERSUS TIME

OPERATION	SIZE	START TIME	DATA AVAILABLE
1. $QG^T$	$(pxp)(pxn)$	1	$n + p + 1$
2a. $b = PH^T$	$(nxn)(nxm)$	$n + p + 1$	$3n + p + 1$
2b. $PF^T$	$(nxn)(nxn)$	$2n + p + m + 1$	$3n + p + m + 1$
3. $R + Hb$	$(mxn)(nxm)$	$3n + p + m + 1$	$4n + p + 2m + 1$
4. $G(QG^T)$	$(nxp)(pxn)$	$3n + p + 3m + 1$	$5n + p + 3m + 1$
5. $(R + Hb)^{-1}$	$(mxm)$		
5a. UPPER DECOMPOSITION.	$(mxm)$	$5n + p + 3m + 1$	$5n + p + 4m + 1$
5b. UPPER INVERSE	$(mxm)$	$5n + p + 4m + 1$	$5n + p + 5m + 1$
5c. $U^{-1}(U^{-1})^T$	$(mxm)(mxn)$	$5n + p + 5m + 1$	$6n + p + 5m + 1$
6a. $a = FK$	$(nxn)(nxm)$	$5n + p + 6m + 1$	$7n + p + 6m + 1$
6b. $Fx$	$(nxn)(nx1)$	$6n + p + 7m + 1$	$7n + p + 7m + 1$
7. $K = b(R + Hb)^{-1}$	$(nxm)(mxm)$	$6n + p + 7m + 2$	$8n + p + 7m + 2$
8. $z - Hx$	$(mxn)(nx1)$	$7n + p + 8m + 2$	$8n + p + 9m + 2$
9a. $F - aH$	$(nxm)(mxn)$	$7n + p + 9m + 3$	$9n + p + 9m + 3$
9b. $x = Fx + a(z - Hx)$	$(nxm)(mx1)$	$9n + p + 9m + 3$	$10n + p + 9m + 3$
10. $P = GQG^T + (F - aH)(PF^T)$	$(nxn)(nxn)$	$9n + p + 9m + 4$	$11n + p + 9m + 4$

Kalman matrix dimensions,  $m$ ,  $n$ , and  $p$  where  $n \geq p \geq m$ . It corresponds to the time when the first element is input to  $PE_{11}$ . The data available time corresponds to the first computational cycle when a resultant element is available at the output of the array.

Table 7 reflects the operational order derived by Papadourakis and Taylor for the special case  $n = m = p$  (1986). Equations (6a) and (6b) of the table have been swapped with equation (7) to ensure that the inverse will be processed and output before the operation of equation (7). Multiplexer switches could be used here, but they are not necessary.

Note from Table 7 that  $b$  is available before it is needed for the  $R + Hb$  calculation. Similarly,  $QG^T$  is available before the  $G(QG^T)$  calculation commences,  $a$  is calculated before the  $F - aH$  calculation requires it, and  $F - aH$  is output prior to the calculation of  $P$ .

The inversion is a special case. The calculation of  $R + Hb$  is completed well in advance of the start of the inverse operation. The upper decomposition and the inverse of the upper triangular matrix operations are each calculated in  $m$  computational units. However, data is available for each operation  $n$  computational units after it starts. If  $n$  is significantly greater than  $m$ , then much time will be spent waiting for the necessary data to start the upper triangular inverse calculation. For example, if  $n = 5$  and  $m = 2$ , three computational units will be wasted. In order to avoid this problem, a switch should be included after  $PE_{mi}$  for  $i$  from one to  $m$ . For the upper triangular decomposition, this switch should reroute data from the top of the left-most  $m$  processors of row  $m$  to the bottom of the array. For the upper triangular inversion, the switch should reroute data from the right of the lower  $m$  processors of column  $m$  to the bottom of the array. This will improve the computational throughput by  $2(n - m)$  computational units. Although this switching breaks the systolic flow of the data, it can, in certain applications, produce a significant performance improvement.



### Register Level Simulation

A C simulation of the systolic array architecture was written to verify the algorithms and architecture proposed for performing the Kalman filter operations. This program was written at a register level and modelled the PE with a LNS based ALU. The simulation was somewhat crude in that it did not handle zero numbers, it did not propagate overflow flags, and it did not implement the transpose switch.

It was assumed that a PE computational unit corresponded to one clock period. Further, it was assumed that  $n = 5$ ,  $p = 3$ , and  $m = 2$ . The intent of the simulation was to model the PE to the 20-bit accuracy described in the previous section. With this model, the performance of the wavefront control structure could be tested along with the load, multiply, and inverse algorithms. Since the nature of the operations in Table 7 are very repetitive, it is sufficient to simulate each case once.

The most encompassing operation is the inverse operation  $(R + Hb)^{-1}$ . This operation requires decomposing a matrix to an upper triangular form, inverting the upper triangular matrix, inputting a matrix transposed, and multiplying two matrices. Fortunately, a very simple example can be applied that will completely test the complex procedures associated with the orthogonal array. Let

$$R + Hb = \begin{bmatrix} 1 & 2 \\ 2 & 5 \end{bmatrix}. \quad (44)$$

Note that this matrix is symmetric. This matrix is decomposed to an upper triangular matrix in 2 clocks. The result is

$$U = \begin{bmatrix} 1 & 2 \\ 0 & 1 \end{bmatrix}. \quad (45)$$

The upper triangular matrix is inverted in 2 clocks. Its inverse is

$$V = \begin{bmatrix} 1 & -2 \\ 0 & 1 \end{bmatrix}. \quad (46)$$

Equations (45) and (46) can be verified with equations (23) through (26). Figure 14 shows the data flow through the systolic array for equations (45) and (46). The inverse matrix is loaded in the array as a result of the previous operation. The transposed matrix input and the matrix multiplication starts immediately after the inverse matrix is output since processing is pipelined. The multiplication is complete after 2 clocks. The multiplication produces

$$V V^T = \begin{bmatrix} 5 & -2 \\ -2 & 1 \end{bmatrix}. \quad (47)$$

which is the inverse of equation (44). The data flow for this step is depicted in Figure 15. The upper decomposition procedure requires that each row of PEs be programmed uniquely. Furthermore, the upper triangular matrix inversion procedure requires that each column of PEs be programmed uniquely. Therefore, each PE must be programmed uniquely, depending upon its position in the array.

The upper control word for  $PE_{11}$  is shown below for the six clock intervals associated with the  $2 \times 2$  inversion procedure. The control words for  $PE_{12}$  are similar except the storage of  $v_{12}$  occurs one clock later than for  $v_{11}$  (relative to the first opcode received by the particular PE). Also,  $PE_{11}$  is a diagonal element and requires slightly different arithmetic functionality for the upper decomposition operations. The control words for  $PE_{21}$  are similar except the storage of  $u_{12}$  occurs one clock later than for  $u_{11}$ .



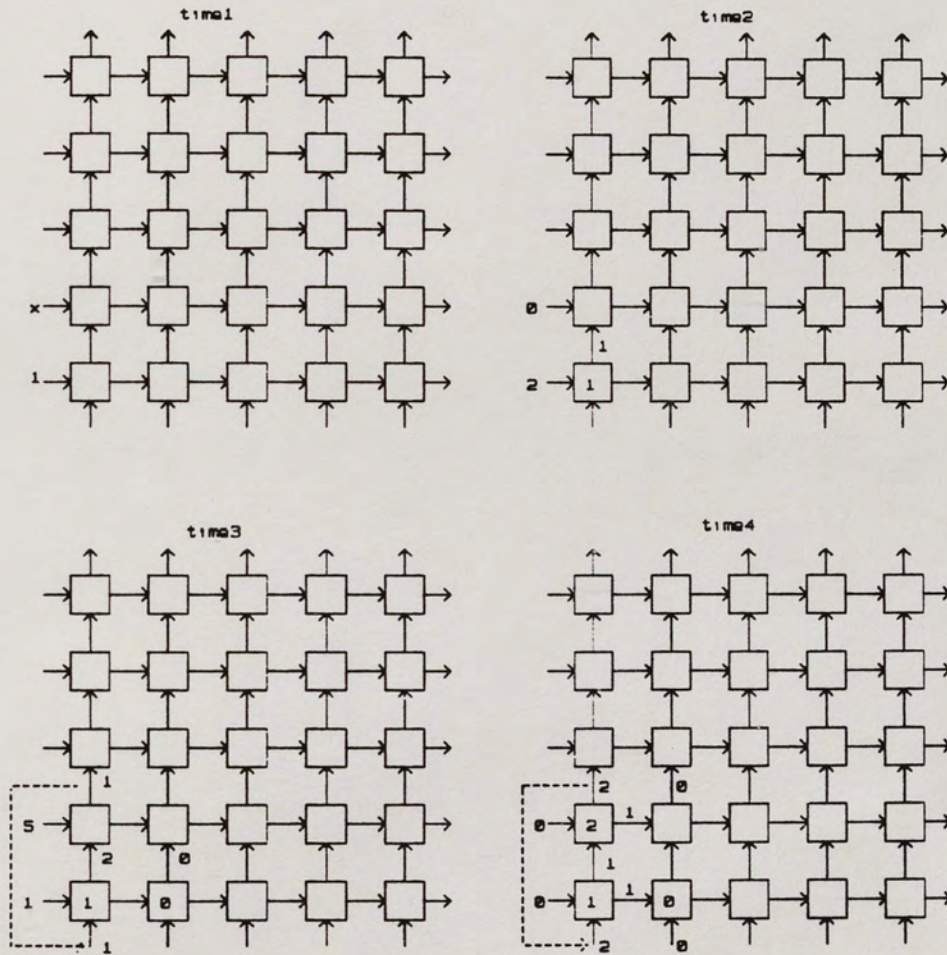


Figure 14. Data Flow For Decomposition and Inverse Operations.

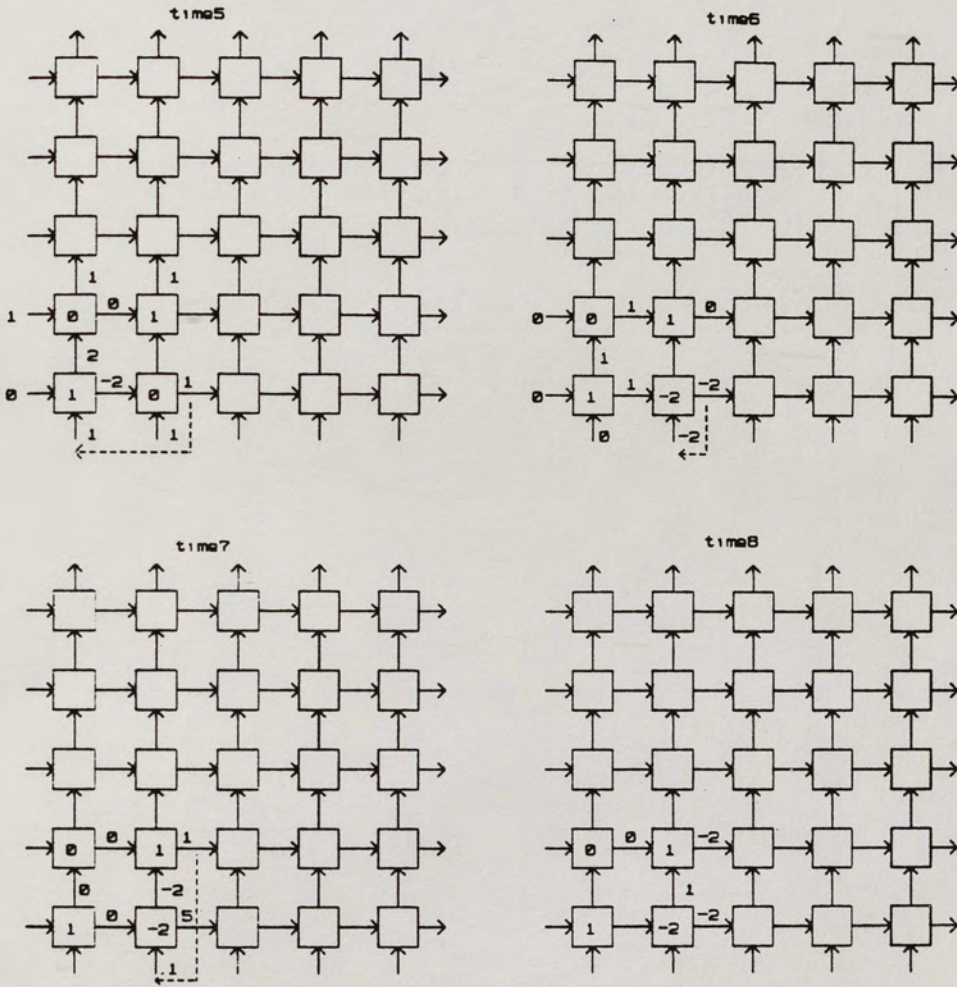


Figure 15. Data Flow For Multiplication Operation.



	31	30	28	27	25	24	22	21	19	18	16
TIME	UNUSED	ALU	MULT_X	MULT_Y	SQUARE_X	ADD_X					
1	0	000	000	000	000	000					
2	0	000	000	000	101	000					
3	0	000	000	000	001	000					
4	0	100	101	101	001	000					
5	0	101	101	101	001	000					
6	0	101	101	101	001	000					

The lower control word is shown below for  $PE_{11}$ .

	15	13	12	10	9	7	6	4	3	2	0
TIME	ADD_Y	TOP	RIGHT	MEM	W/R	MEM_ADDR					
1	000	011	001	011	1	010					
2	000	010	001	000	0	010					
3	000	000	010	010	1	011					
4	010	000	100	000	0	011					
5	010	000	100	000	0	011					
6	010	000	100	000	0	011					

### Performance Comparison

The performance of the new Kalman filter implementation is compared to two previous designs in Table 8. This table is a modified version of one developed by Papadourakis and Taylor (1986). The first design uses algorithms developed by Liu and Young (1984). The second design was developed by Kung (1982). The comparison is not exactly apples to apples. However, letting  $n = p = m$  produces a total computational throughput of  $21n + 3$  for the new method. The comparison of the different approaches for various values of  $n$  is shown in Table 9.

TABLE 8  
KALMAN FILTER EXECUTION TIME COMPARISON

OPERATION	SIZE	BAPST	LIU	KUNG
		PAPADOURAKIS	YOUNG	
1. $QG^T$	$(pxp)(pxn)$	$n + p$	$4n - 2$	$4n - 1$
2a. $PH^T$	$(nxn)(nxm)$	$n + m$	$4n - 2$	$4n - 1$
2b. $PF^T$	$(nxn)(nxn)$	$n$	$n$	$4n - 1$
3. $R + Hb$	$(mxn)(nxm)$	$2m$	$4n - 2$	$5n - 1$
4. $G(QG^T)$	$(nxp)(pxn)$	$2n$	$4n - 2$	$4n - 1$
5. $(R + Hb)^{-1}$	$(mxm)$	$3m$	$32n - 16$	$36n - 8$
6a. $FK$	$(nxn)(nxm)$	$n + m$	$4n - 2$	$4n - 1$
6b. $Fx$	$(nxn)(nx1)$	1	1	$4n - 1$
7. $b(R + Hb)^{-1}$	$(nxm)(mxm)$	$n + m$	$4n - 2$	$4n - 1$
8. $z - Hx$	$(mxn)(nx1)$	$m + 1$	$3n - 1$	$3n + 1$
9a. $F - aH$	$(nxm)(mxn)$	$2n$	$4n - 2$	$5n - 1$
9b. $Fx + a(z - Hx)$	$(nxm)(mx1)$	1	1	$3n + 1$
10. $GQG^T + (F - aH)(PF^T)$	$(nxn)(nxn)$	$2n$	$4n - 2$	$5n - 1$
	TOTAL	$11n + 9m + p + 3$	$68n - 31$	$85n - 16$

TABLE 9  
COMPARISON OF KALMAN FILTER IMPLEMENTATIONS VERSUS N

N	BAPST	LIU	KUNG
	PAPADOURAKIS	YOUNG	
3	66	173	239
5	108	309	409
10	213	649	834
64	1347	4321	5424
100	2103	6769	8484

Clearly, the new implementation becomes even more favorable as  $n$  increases. The time to execute the Kalman filter equations is about three to four times faster than either of the previous methods.



PE utilization is a measure of the average number of PEs used in a computational unit of time. For the Kalman filter implementation, the total number of PEs used for each operation is summarized in Table 10. As shown, the total number of PEs used for an  $n \times n$  multiply is  $n^2(n + 1)$ . Actually,  $n^2$  processors are used during the load and  $n^3$  are used for the actual multiply. Given the equations of Table 10, the PE utilization is

$$PEU = \frac{4n^3 + 6n^2 + np^2 + p^2 + np + 3mn^2 + 2nm^2 + m^2 + 4mn + 2m^3}{11n + 9m + p + 3}. \quad (48)$$

The denominator, of course, is the total Kalman filter execution time defined in Table

TABLE 10

## PE UTILIZATION OF KALMAN FILTER IMPLEMENTATION

OPERATION	SIZE	TOTAL PEs USED
1. $QG^T$	$(pxp)(pxn)$	$p^2(n + 1)$
2a. $PH^T$	$(nxn)(nxm)$	$n^2(m + 1)$
2b. $PF^T$	$(nxn)(nxn)$	$n^3$
3. $R + Hb$	$(mxn)(nxm)$	$mn(m + 1)$
4. $G(QG^T)$	$(nxp)(pxn)$	$n(n^2 + p)$
5. $(R + Hb)^{-1}$	$(mxm)$	
5a. UPPER DECOMPOSITION.	$(mxm)$	$m^3$
5b. UPPER INVERSE	$(mxm)$	$m^3$
5c. $U^{-1}(U^{-1})^T$	$(mxm)(mxm)$	$m^2(n + 1)$
6a. $FK$	$(nxn)(nxm)$	$n^2(m + 1)$
6b. $Fx$	$(nxn)(nx1)$	$n^2$
7. $b(R + Hb)^{-1}$	$(nxm)(mxm)$	$nm(n + 1)$
8. $z - Hx$	$(mxn)(nx1)$	$n(n + m)$
9a. $F - aH$	$(nxm)(mxn)$	$n(n^2 + m)$
9b. $Fx + a(z - Hx)$	$(nxm)(mx1)$	$n^2$
10. $GQG^T + (F - aH)(PF^T)$	$(nxn)(nxn)$	$n^2(n + 1)$

8. For  $n = m = p = 5$ , the PE utilization is  $1825/108 = 16.9$ . Therefore,  $16.9/25 = 67.6$  percent of the PEs are being used on the average. If  $n = 5$ ,  $p = 3$ , and  $m = 2$ , the PE utilization is  $969/79 = 12.3$ , so 49.1 percent of the PEs are used on the average. Clearly, the PE utilization will decrease as  $m$  and  $p$  differ from  $n$ .

Other performance measurements are available for advanced parallel systems. The Optimum Processor Count (OPC) reflects the basic parallelism within the system. It is the number of PEs needed to handle a given array size. For the Kalman filter application, it is  $n^2$ .

The speedup (SU) is defined as the ratio of serial time to parallel time. A matrix multiply requires  $n^3$  serial multiply-accumulate type operations. Assuming that all fifteen basic Kalman operations are of this type and  $n = m = p$ , the speedup for the special Kalman filter implementation is

$$SU = \frac{12n^3 + 3n^2}{21n + 3}. \quad (49)$$

For  $n = 5$  the speedup is 14.6.

It was assumed that the serial type computer used for the preceding comparison employed an LNS ALU. A floating point ALU such as the AM29325 would require three computational units to perform one calculation since time is needed for normalizing and denormalizing data before arithmetic operations. Therefore, the speedup would be three times that mentioned above for a floating point ALU.



## CHAPTER IV, CONCLUSION

An implementation of the recursive Kalman filter was obtained using a systolic architecture. A few instruction multiple data machine was used to perform the individual operations such as matrix multiplication and inversion. The parallel processor was reconfigurable which allowed all operations to be calculated in a strict pipeline fashion with one array of processing elements. The algorithms developed were general in terms of matrix operations and the Kalman filter array dimensions.

The throughput of this new design exceeded previous implementations employing parallel processors by a factor of three to four depending on the Kalman dimensions. For  $n = 5$ , the new implementation is roughly 14 times faster than a LNS based serial processor implementation, and the processing element utilization is about 67 percent.

A LNS based processing element was used to provide a machine with a capability for a large dynamic range. The resulting array, which was proven and verified with a C program, was completely designed. Details such as PE control were included in this paper. The result is not only general, but it is also fairly simple to understand and implement. The technically limiting factor will be in the physical implementation of the processing element. A small feature size VLSI process will be necessary if such a design is to have a one clock computational time. Although the LNS ALU allows functions such as division and square root to be calculated easily and with minimum

hardware, driving off one VLSI part and onto another has proven to be a speed critical problem at high frequencies.

### Alternate Architectures

Other computer architectures could have been used to implement the Kalman filter, but it is not obvious that any could improve upon the performance obtained here. A serial computer would produce the most hardware efficient design, but it would suffer from obvious computational bandwidth problems. A vector processor would improve upon the serial computer's performance, but would still suffer from bandwidth limitations. An orthogonal architecture similar to the one developed here could use fixed point or floating point processing elements. Either of these approaches would have a difficult time performing the square root and division operations necessary for the inversion algorithm. To revert to the iterative inversion algorithm would result in reduced performance.

### Areas of Future Work

A compiler is the next logical step for work in this area. To completely generalize this design, a compiler would be necessary to transform any given operation or set of operations into an efficient and logical series of pipelined systolic operations. Combined with this effort, the firmware of the general implementation should be written for the PEs and external control.



The handling of some special LNS conditions such as zero and overflow should be given more thought. Such items would involve more detail than was appropriate for this paper. However, zero data appears frequently and the handling of such data should be considered.

The memory and interface structures should be considered in more detail. It was mentioned briefly that a four port random access memory could serve each column of PEs. This memory, which is certainly feasible in a register file implementation, should have two read and two write ports. This would allow it to input to a left side element and a bottom side element simultaneously. In addition, it could store data that is output from a top column element and a right row element simultaneously. The transpose switch could be used to "shuffle" the data as it is input to the array.

## REFERENCES

- Briggs, Faye A., and Hwang, Kai. Computer Architecture and Parallel Processing. New York: McGraw-Hill Book Company, 1984.
- Graham, James H., and Kadela, Thaddeus F., "Parallel Algorithms and Architectures for Optimal State Estimation," IEEE Transactions Computers, Vol. C-34, November 1985, pp. 1061-1068.
- Kalman, R.E., "A New Approach to Linear Filtering and Prediction Problems," Journal of Basic Engineering, Vol. 82, March 1960, pp. 35-45.
- Kung, H. T. "Why Systolic Architectures?" Computer, January 1982, pp. 37-46.
- Liu, Philip S., and Young, Tzay Y., "VLSI Array Design Under Constraint of Limited I/O Bandwidth," IEEE Transactions Computers, Vol. C-32, December 1984, pp. 1160-1170.
- Papadourakis, George M., "Adaptive Optimal Filtering using the Logarithmic Number System." Ph.D. dissertation, University of Florida, 1986.
- Papadourakis, George M., and Taylor, Fred J., "Implementation of Kalman Filter using Systolic Arrays," University of Central Florida and University of Florida, 1986.
- Pucknell, Douglas A., and Eshraghian, Kamran, Basic VLSI Design, Sydney: Prentice-Hall, 1985.
- Stone, Harold S., Introduction to Computer Architecture, Chicago: Science Research Associates, 1980.
- Taylor, Fred J., "An Extended Precision Logarithmic Number System," IEEE Transactions ASSP, Vol. ASSP-31, February 1983, pp. 232-234.
- Taylor, Fred J., "A Hybrid Floating-Point Logarithmic Number System Processor," IEEE Transactions Circuits and Systems, Vol. CAS-32, January 1985, pp. 92-95.



Taylor, Fred J., "A 20-bit VLSI Arithmetic Unit for Digital Signal Processing in the Logarithmic Number System," Signal Processing III; Theories and Application, Elsevier Publishers, 1987.