

Retrospective Theses and Dissertations

1987

Design of a Processing Element for the Systolic Array Implementation of a Kalaman Filer

John P. Condorodis
University of Central Florida

 Part of the [Computer 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.

STARS Citation

Condorodis, John P., "Design of a Processing Element for the Systolic Array Implementation of a Kalaman Filer" (1987). *Retrospective Theses and Dissertations*. 5028.
<https://stars.library.ucf.edu/rtd/5028>

THE DESIGN OF A PROCESSING ELEMENT FOR THE SYSTOLIC
ARRAY IMPLEMENTATION OF A KALMAN FILTER

BY

JOHN P. CONDORODIS
B.E.E., Georgia Institute of Technology, 1984

RESEARCH PAPER

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

Spring Term
1987

ABSTRACT

The Kalman filter is an important component of optimal estimation theory. It has applications in a wide range of high performance control systems including navigational, fire control, and targeting systems. The Kalman filter, however, has not been utilized to its full potential due to the limitations of its inherent computational intensiveness which requires "off-line" processing or allows only low bandwidth real-time applications.

The recent advances in VLSI circuit technology have created the opportunity to design algorithms and data structures for direct implementation in integrated circuits. A systolic architecture is a concept which allows the construction of massively parallel systems in integrated circuits and has been utilized as a means of achieving high data rates. A systolic system consists of a set of interconnected processing elements, each capable of performing some simple operation.

The design of a processing element in an orthogonal systolic architecture will be investigated using the state of the art in VLSI technology. The goal is to create a high speed, high precision processing element which is adaptive to a highly configurable systolic architecture. In order to achieve the necessary high computational throughput, the arithmetic unit of the processing element will be implemented using the Logarithmic Number System. The Systolic architecture approach will be used in an attempt to implement a Kalman filtering system with both a high sampling rate and a

small package size. The design of such a Kalman filter would enable this filtering technology to be applied to the areas of process control, computer vision, and robotics.

ACKNOWLEDGMENTS

I would like to thank my committee chairman, Dr. George M. Papadourakis, for his valuable guidance and assistance in the writing of this research paper. In the past months he has provided me with a great deal of support and encouragement which helped in the completion of my masters.

I would also like to thank Dr. Christian S. Bauer, Dr Darrell G. Linton, and Tom R. Hoffman for serving on my committee and providing me with many useful comments.

Finally, I would like to thank my parents, my brother, and my sister for their support, encouragement, and a great deal of understanding which they have given me during the completion of my masters.

TABLE OF CONTENTS

LIST OF TABLES	vi
LIST OF FIGURES	vii
CHAPTER I, INTRODUCTION	1
The Kalman Filter	1
The Systolic Architecture	2
The Processing Element	4
CHAPTER II, PRELIMINARY DESIGN INFORMATION	6
Kalman Filter Equations	6
Systolic Architectures	9
Proposed Systolic Architecture	13
Equations for Proposed Architecture	15
Matrix Operations	16
VLSI Technology	17
Logarithmic Number System	18
Floating Point to LNS Conversion	20
LNS to Floating-Point Conversion	22
Conversion Chip	24
CHAPTER III, PROCESSING ELEMENT DESIGN	27
The LNS ALU	27
Description of Arithmetic Algorithms	29
Memory Reduction Technique	33
ALU Components	35
ALU Data Paths	37
ALU Layout and Timing	42
PE Interface Structure	46
Internal Interfaces	46
External Interface	48
PE Control Structure	51

Counter/Comparator Block	51
Control Words	52
Memory Organization	54
External Control	56
Self Test	57
Programming the PE Microcode Controller	57
System Clock	58
PE Simulation	59
Chip Layout and Key Parameters	65
Chip I/O	65
Chip Size and Speed	66
 CHAPTER IV, CONCLUSION	 69
Alternative PE Designs	69
Areas of Future Research	70
 REFERENCES	 72

LIST OF TABLES

1. Filter Variables	8
2. Geometric Configurations and Corresponding Functions	11
3. The Required Functions of the Systolic Architecture	16
4. Conversion Chip Speed Estimate	26
5. Logic for the Addition and Subtraction Functions	32
6. Memory Requirements for One LOG Function	35
7. ALU Component and Function Timing	44
8. OP-CODES for Internal Bus Switches	47
9. OP-CODES for Memory Bus Switches	48
10. OP-CODES for External Bus Switches	49
11. OP-CODES for PE Control	56
12. Chip I/O Description	66

LIST OF FIGURES

1. Conventional vs. Systolic Processor Architectures	10
2. Typical Systolic Architecture Configurations	12
3. Systolic Pipelined Orthogonal Array	14
4. Floating Point to LNS Conversion Block Diagram	21
5. LNS to FLP Conversion Algorithm Block Diagram	23
6. Block Diagram of Conversion Chip Layout	25
7. Block Diagram of Proposed PE	28
8. Multiplication/Division Data Path	38
9. Addition/Subtraction Data Path	40
10. Square/Square Root Data Path	41
11. ALU Block Diagram	43
12. ALU Floorplan	45
13. Block Diagram of PE Bus Structure	50
14. Block Diagram of PE Control Structure	55
15. Flow Diagram of First Microcode Function	62
16. Flow Diagram for Second Microcode Function	64
17. Chip Layout	68

CHAPTER I, INTRODUCTION

The Kalman filter is an important tool used in many modern control systems. However, the amount of intensive calculations required to implement the filter have limited the Kalman filter's applications. Systolic architectures offer a method of increasing the speed of the matrix arithmetic required in the construction of a Kalman filter. The current state of the art in very large scale integrated (VLSI) circuits has made feasible the design of high performance computational systems on a single silicon chip. The design of a processing element utilizing VLSI technology should allow the construction of a systolic architecture capable of implementing a Kalman filter.

The Kalman Filter

Measuring the state variables of a system for use in the control of the system is a fairly simple idea; however, this task is complicated if noise is present in the system. The presence of noise in the system requires the estimation of states (Graham and Kadela 1985). The Kalman filter is one of the most popular state estimators used in system control and it is well-suited to the problem of monitoring measurement consistency and the related statistics (Papadourakis and Taylor 1986).

In 1960 Kalman developed the linear estimation theory that is now referred to as the Kalman filter (Kalman 1960). The theory was developed further over time to give more insight into the linear estimation problem. The earliest applications of the

Kalman filter dealt with satellite orbit determination, tracking, and navigation problems. The filter has also found applications in the areas of anti-aircraft gun fire control, ship navigational control, and seismic data processing in oil exploration (Papadourakis 1986).

Unfortunately, the intensive matrix calculations required to implement the Kalman filter have limited its applications. For example, many real-time applications exceed the throughput capability of the conventionally designed Kalman filter. A possible method of solving the required calculations at a rate applicable to real-time processing is to use a parallel processing architecture. The systolic architecture's approach to parallel processing offers a feasible method of performing the high speed matrix multiplications necessary for a Kalman filter. A possible implementation of the systolic arrays approach was presented by Graham and Kadela who have summarized the necessary matrix calculations necessary to implement the filter (Graham and Kadela 1985).

The Systolic Architecture

Systolic architectures offer a method of optimizing an algorithm for direct layout in integrated circuits. The systolic architectural concept was developed by Kung and associates at Carnegie-Mellon University. In a systolic system, data flows from the computer memory in a rhythmic fashion, passing through many processing elements before it returns to memory, much as blood circulates to and from the heart. Many versions of systolic processors are currently being designed by various universities and

industrial organizations which will result in cost-effective, high-performance, special-purpose systems for a wide range of potential applications (Briggs and Hwang 1984).

A systolic array architecture consists of a network of processors which communicate with their neighboring processors. The processing elements of a systolic array may be arranged in several geometric configurations which allow it to perform different functions. Systolic arrays are configurable for a wide range of applications including matrix multiplication, fast Fourier transforms, and pattern matching.

The basic principle of a systolic system is quite simple. Replacing a single processing element with an array of processing elements will result in a higher computational throughput without an increase in memory bandwidth (Kung 1982). The method used to implement this principle is to make exhaustive use of data brought out of memory before it is returned to memory. The information in a systolic system flows rhythmically between neighboring elements in a pipelined method. The data from memory passes through the systolic system, making it available for each of the array's processing elements.

Systolic architectures can be used to speed up compute-bound computation in a relatively simple and inexpensive manner. The systolic array architecture thus offers a viable method of constructing a low-cost compact processor capable of performing the calculations required to implement a Kalman filter. To construct the desired systolic array it would be necessary to design a processing element capable of high speed computations and data routing between neighboring processing elements.

The Processing Element

The processing element (PE) is essentially an arithmetic logic unit (ALU) with attached working registers and local memory (Briggs and Hwang 1984). The PE is normally designed to perform some simple operations along with the ability to network to other PEs.

The construction of a PE capable of high speed calculations and data routing between neighboring elements will require a departure from the normal PE design. The ALU must be able to perform multiplication, division, addition, and subtraction at a very high speed. The PE must also be capable of shifting data to neighboring PEs to allow the configuration of various algorithms. The first step in the design of a suitable PE will be the determination of an acceptable numbering system for use in the construction of the ALU.

The Logarithmic Number System (LNS) has been extensively studied by various authors with regards to applications in the digital signal processing environment. In particular, Kurokawa demonstrated that LNS gives superior filtering performance compared to that of floating point systems of equivalent word length and range (Kurokawa 1980). The numbers in LNS are a signed radix raised to some signed exponent. The multiplication and division operations are simply an addition or subtraction of the exponents.

LNS offers the advantage of providing a considerable improvement in multiplication and division performance over other numbering systems. The major disadvantage of LNS is that the addition and subtraction operations require the use of a memory look-up table. The memory required to implement an LNS based ALU has been found to increase drastically as word length is enlarged. The amount of memory required to utilize large word lengths of 16 bits or more have limited the practical applications of LNS. However, the current advances in VLSI technology have made feasible the construction of a 20 bit ALU based on LNS. A LNS based ALU will have the speed and accuracy required to design an advanced PE capable of performing Kalman filtering calculations.

CHAPTER II, PRELIMINARY DESIGN INFORMATION

In order to construct a high speed processing element capable of being used to implement a Kalman filter it is necessary to have some preliminary information. The designer must have an understanding of the Kalman filter matrix equations, the type of systolic architecture that will be utilized, the VLSI technology that will be used to produce the design, and the Logarithmic Number system (LNS).

Kalman Filter Equations

The application of optimal estimation is predicated on the description of a physical system under consideration by means of mathematical models. The early work in the area of control and estimation theory involved system description and analysis in the frequency domain. In early 1960 the system description was made in the time domain using state-space notation which offers the advantage of mathematical and notational convenience. This method also has the advantage of producing a system description closer to physical reality than any of the frequency oriented techniques. The time domain approach is particularly useful in providing a statistical description of the system behavior. Initially, the work done in the time domain was concerned with continuous systems, but recently work has been extended to the discrete case (Papadourakis and Taylor 1985). The Kalman filter is one of the most popular state estimators used in system control which utilizes the discrete case models.

The availability of discrete case models has allowed the use of digital computers to simulate the system. However, even though the equations are straightforward in solution their are computationally intensive. Many authors have studied methods of simplifying the matrix computations required to implement the Kalman filter. In particular Graham and Kadela have summarized the matrix equations necessary to perform Kalman filtering. The equations developed by Graham and Kadela where also shown to reduce the amount of matrix multiplication by 25% (Graham and Kadela 1985).

A discrete dynamic system can be represented as the following equations,

$$x(k+1) = F(k)*x(k) + G(k)*w(k) \quad (1)$$

and

$$z(k) = H(k)*x(k) + v(k) \quad (2)$$

where the variables are defined in Table 1. The random variables $x(0)$, v , and w have known covariance statistics $P(0)$, R , and Q . The problem is to find the best estimate \hat{x} for x given m measurements such that the sum of the squares of the estimation is a minimum.

For the time-invariant system, the estimate at time k is defined by

$$\hat{x}(k + 1 | k) = \hat{x}(k | k - 1) + K*(z(k) - H*\hat{x}(k | k - 1)) \quad (3)$$

where K is a gain or weighting matrix that meets the minimization criteria. A new update is determined by using the state transition matrix to get

TABLE 1
FILTER VARIABLES

VARIABLE	IDENTITY	SIZE
k	0,1,2...	1x1
x	state estimate vector	nx1
P	state estimate covariance matrix	nxn
F	state transition matrix	nxn
G	system disturbance distribution	nxs
Q	system disturbance covariance	sxs
H	measurement matrix	rxn
R	measurement noise covariance	rxr
K	gain matrix	nxr
v	plant noise (white Gaussian)	rx1
w	system noise (white Gaussian)	sx1
z	measurement vector	rx1

$$\hat{x}(k+1|k) = F*\hat{x}(k|k-1) + F*K*(z(k) - H*\hat{x}(k|k-1)). \quad (4)$$

The error covariance for the updated estimate is given by

$$P(k+1|k) = F*P(k|k)*F^T + G*Q*G^T, \quad (5)$$

and it can be shown that

$$P(k|k) = P(k|k-1) - K*H*P(k|k-1). \quad (6)$$

Equations (5) and (6) can be merged together to yield

$$P(k) = F*P(k-1)*F^T - F*K*H*P(k-1)*F^T + G*Q*G^T. \quad (7)$$

The gain matrix K is found by taking the partial derivative of the sum of the squares of the estimation error with respect to K and setting it equal to zero. The resulting

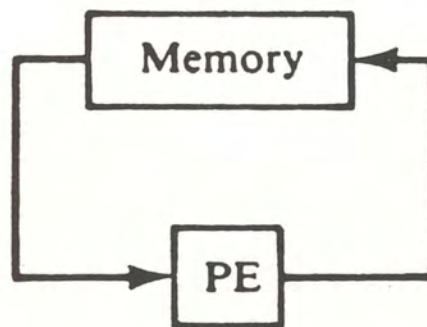
solution for K is

$$K = P(k - 1) * H^T * (H * P(k - 1) * H^T + R)^{-1}. \quad (8)$$

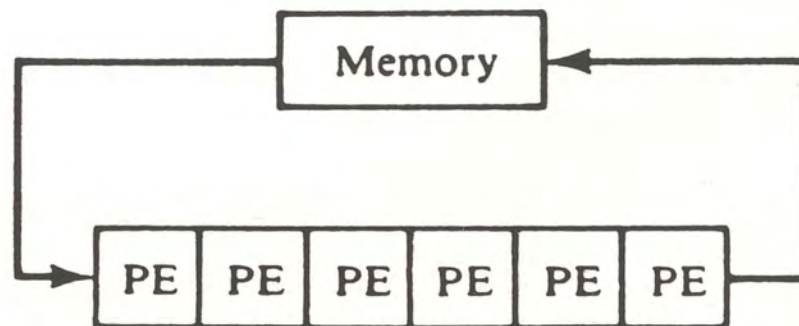
Equations (4), (7), and (8) define the recursive optimal estimation filter. However, the recursion must first be initialized by estimating $x(0)$ and $P(0)$. This produces the first estimate as $x(k + 1 | k) = F * x(k | k)$ or $x(1 | 0) = F * x(0)$. Once the first measurement is completed, equations (4), (7), and (8) are considered valid and may be used to implement a Kalman filter.

Systolic Architectures

A systolic structure consists of many processing elements connected to their neighboring processing elements in a regular fashion. The crux of the systolic architecture approach is to ensure that once a data item is brought out of memory it can be used effectively at each cell it passes (Manno 1986). Figure 1 shows the difference between a conventional processor and a systolic array processor. Suppose each PE in Figure 1 operates with a clock rate of 100 ns. The conventional memory-processor organization has at most a performance of 5 million operations per second. With the same clock rate, the systolic array will result in a performance of 30 million operations per second. The ability to use each data item a number of times is only one advantage of the systolic architecture. Other advantages include modular expansion, regular data flow, and use of uniform processing elements.



(a) The conventional processor



(b) A systolic processor array

Figure 1. Conventional vs. Systolic Processor Architectures.

The systolic architecture can be designed to implement different algorithms by using various geometric configurations. A systolic architecture can also be designed to allow reconfigurability for different algorithms, so that only one implementation need be constructed. Figure 2 illustrates a few typical configurations of systolic architectures. A summary of the computational functions performed by the displayed configurations can be seen in Table 2.

TABLE 2

GEOMETRIC CONFIGURATIONS AND CORRESPONDING FUNCTIONS

PROCESSOR ARRAY STRUCTURE	COMPUTATIONAL FUNCTION
linear arrays	Fourier transform, sorts, priority queue
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

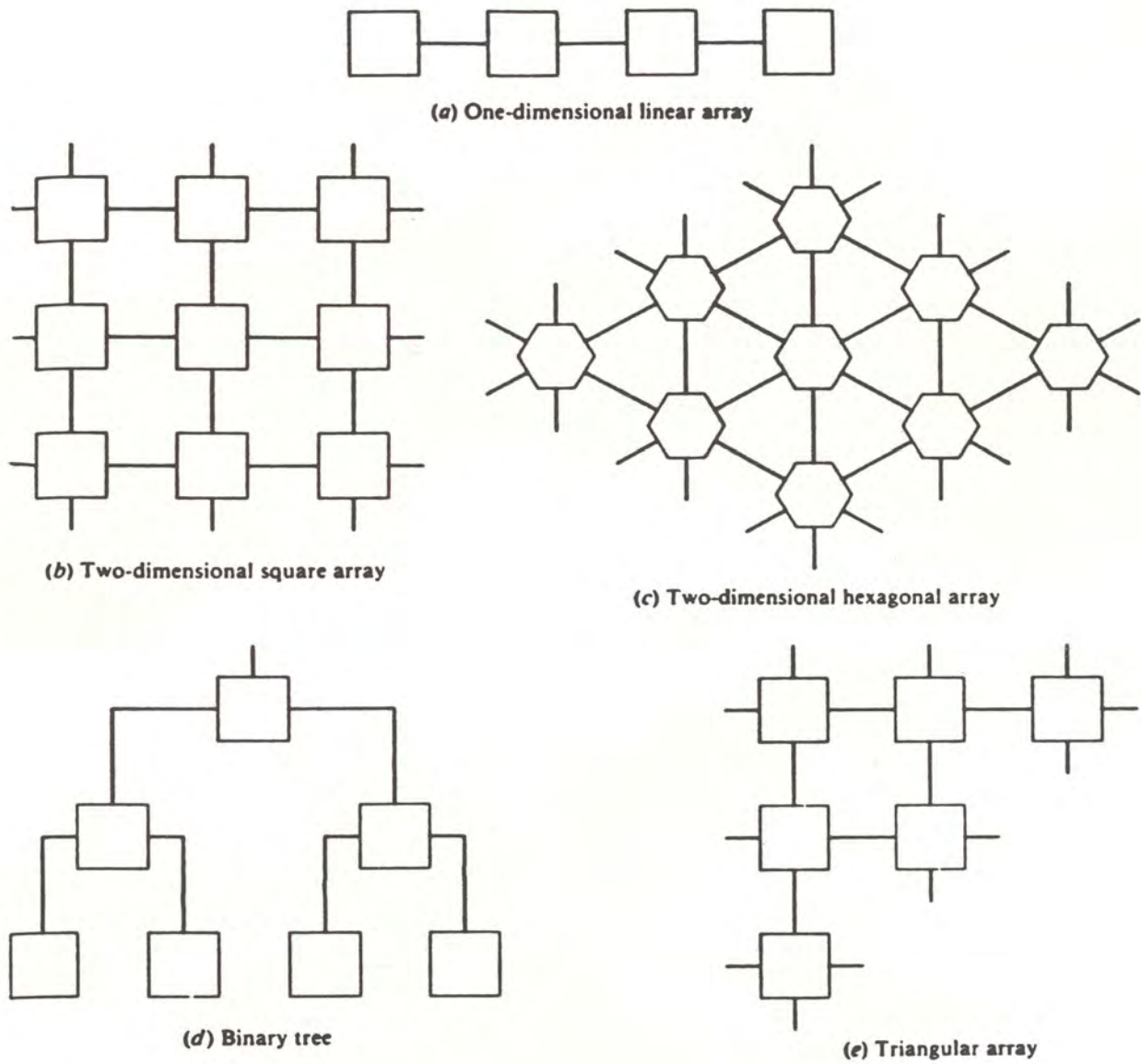


Figure 2. Typical Systolic Architecture Configurations.

Proposed Systolic Architecture

The desired systolic architecture must perform matrix arithmetic operations. The orthogonal array architecture from Table 2 will meet the basic needs for the implementation of the Kalman filter. There are many algorithms available for performing matrix operations using an orthogonal array architecture. A well known matrix multiplication algorithm was developed by Kung to calculate a result in $4n-1$ computational units, where the matrices are $n \times n$ in size (Kung 1982). Until recently Kung's algorithm was one of the fastest available to perform matrix multiplication. However, a new algorithm which uses extensive pipelining was developed by Papadourakis reducing the matrix multiplication time to $2n$ computational units (Papadourakis 1986).

The new algorithm assumes that matrix A is partially loaded inside the array, B is then piped in to interact with A so operations can take place before A is completely loaded as shown in Figure 3. During each computational cycle the cells in the array are able to perform two distinct operations; the multiplication step and the loading operation. The multiplication operation involves each PE taking the sum of partial products from the left neighbor and adding it to its partial product of (ab) before passing the sum to the right neighbor for the next multiplication step. The data stream of B is also piped one row deeper into the array. While some of the PEs execute the multiplication step the loading of the remaining elements of matrix A can be performed simultaneously. This algorithm will allow the multiplication of two $n \times n$

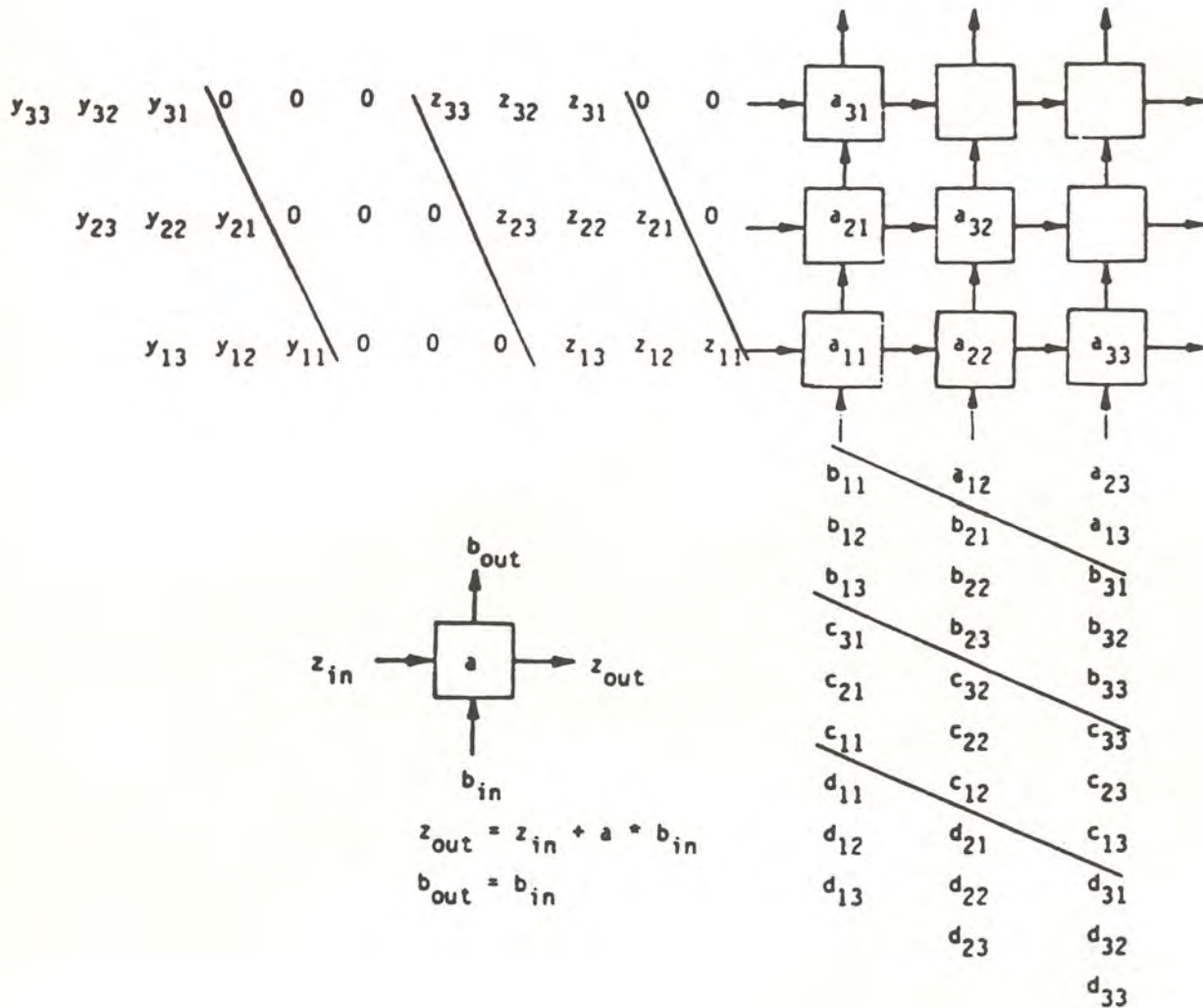


Figure 3. Systolic Pipelined Orthogonal Array.

matrices in a computational time of $2n$ units, compared to $4n-1$ units for Kung's algorithm. The new algorithm will be used along with an orthogonal array architecture to implement the Kalman filter.

Equations for Proposed Architecture

The Kalman filter equations are inherently parallel, but they must be rearranged to take advantage of the pipelined systolic architecture. Equations (4), (7), and (8) can be used in a systolic architecture when expressed as

$$x = F*x + F*K*[z - H*x], \quad (9)$$

$$P = F*P*F^T - F*K*H*P*F^T + G*Q*G^T, \quad (10)$$

and

$$K = P*H^T*[H*P*H^T + R]^{-1}. \quad (11)$$

Graham and Kadela manipulated equations (9), (10), and (12) using commutative, associative, and distributive laws to yield a 25% reduction in the amount of required matrix multiplication. The resulting Kalman filter equations are

$$a = F*K, \quad (13)$$

$$x = F*x + a*[z - H*x], \quad (14)$$

$$P = [F - a*H]*P*F^T + G*Q*G^T, \quad (15)$$

$$b = P*H^T, \quad (16)$$

and

$$K = b*[R + H*b]^{-1}. \quad (17)$$

The above equations can be used in the systolic architecture implementation of a Kalman filter.

Matrix Operations

The Kalman filter equations can be analyzed to determine the matrix operations that must be performed by the systolic architecture. The required functions that the systolic architecture must perform are shown in Table 3.

TABLE 3
THE REQUIRED FUNCTIONS OF THE SYSTOLIC ARCHITECTURE

SYSTOLIC ARRAY FUNCTIONS
<ul style="list-style-type: none"> • Matrix Addition • Matrix Subtraction • Matrix Multiplication • Matrix Transpose • Matrix Inverse

The only operation which is currently difficult to perform using a systolic architecture is the matrix inversion. Graham and Kadela have applied an iterative technique to implement matrix inversion for the Kalman filter. The algorithm is given by

$$D(k+1) = D(k) * (2*I - A * D(k)) \quad (18)$$

where A is the original matrix and D is the resulting inverse (Graham and Kadela 1985). However, this approach requires three to four iterations to arrive at a result. In an attempt to increase the speed of the Kalman filter equations future research should be directed towards deriving a more efficient inversion algorithm. A possibility may be a recently developed algorithm which uses the square root function to perform an

inversion in $(3n-2)$ computational units. This algorithm would greatly increase the speed of the Kalman filter equations, however, it is beyond the scope of this paper to discuss.

VLSI Technology

In order to determine how complex a PE can be designed on a single chip it is important to have an understanding of VLSI technology in general and its relation to the proposed design. Very Large Scale Integration (VLSI) refers to integrated circuits that have a minimum of approximately 10,000 transistors (Abidin, 1984). There are basically three ways to design VLSI circuits; either gate array, standard cell or full custom layouts can be generated. The design of the systolic array PE and all support hardware will be implemented using standard cells. The standard cells approach has been chosen for its advantages of allowing the construction of high gate count circuits and providing short turn around time from design to production.

In order to design devices of VLSI complexity it is necessary to use an integrated computer aided design (CAD) system. The CAD system usually consist of specialized computers and software to perform the necessary tasks that will unburden the designer. The design of the hardware presented in this paper was performed using the VTIttools CAD system from VLSI Technology, Inc. (VTI). The VTIttools system is basically a software package which runs on an Apollo work station. The software used for the PE design was installed on a Domain series 3000 work station. The VTIttools system allowed the standard cells approach to be utilized in the design of the proposed PE.

A major factor in the size of a VLSI chip is the fabrication technology that is used to produce the chip. The technology is defined by the size of the smallest possible transistor with the dimension given in microns. The current state of the art in VLSI technology allows the construction of 2 micron transistors. The PE designed in this paper will use the VTI 2 micron fabrication technology to calculate all size estimations for the proposed circuit. The fabrication technology is critically in the design of an integrated circuit because there is a limit to the chip size which can be produced using current technology. The present limits in VLSI technology will allow the construction of a 400×400 mil chip, where a mil is $1/1000000$ of an inch (Hoffman 1987). At the present time research is being conducted to both reduce the size of transistors below 2 microns and increase total allowable chip size. The introduction of these new technologies would allow larger circuits to be placed on a single chip.

An important point to make is that the designs presented in this paper have only been generated on the VTItools system to produce size and speed estimates. The proposed designs would still require a great deal of test to verify algorithms and circuit layout before a chip could be produced. The VTI system is capable of simulating the proposed PE and this may be an area for future research.

Logarithmic Number System

The Logarithmic Number System (LNS) has been shown to have superior filtering performance compared to a floating-point system of equivalent word length and range. LNS has the ability to perform very fast addition, subtraction, multiplication, division,

squaring, and square roots, along with being easy to implement. The numbers in LNS are represented as a signed radix raised to some sign exponent. However, if the radix is fixed, a number can be uniquely represented by its signed exponent alone. All the arithmetic in the system can be performed using only the exponent (Taylor 1985).

The multiplication and division functions are simply an addition or subtraction operation on the exponents. The calculation of the square and square root for a number consists merely of shifting the exponent left or right respectively by one bit position. An important point to make here is that LNS performs a very quick square root calculation, a function that is critical to a newly developed inversion algorithm. As stated earlier, the new inversion algorithm would significantly increase the speed of the Kalman filter equations. The addition and subtraction operations are more complicated, requiring the use of a look-up table to calculate a Logarithmic base two function. A look-up table is used to compute the base two logarithmic functions because of the availability of efficient and fast memories. The look-up table will be constructed using a Read Only Memory (ROM).

LNS is an ideal numbering system to be used in the implementation of the systolic array Kalman filter. However, to use the LNS approach in the design of an ALU, it would be necessary to convert between LNS and a more acceptable number system. The Floating Point Number System (FLP) is a commonly used number system and will be used to demonstrate the LNS conversion algorithms. The following sections will develop architectures to allow conversions between both 20-bit

LNS numbers and 20-bit FLP (13-bit mantissa; 7-bit exponent) numbers. Block diagrams will also be developed for the conversion algorithms.

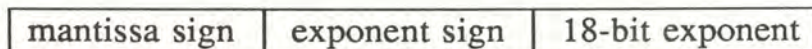
Floating Point to LNS Conversion

A floating point number (FLP) is represented as $m.r^{e_f}$, where m is a 13-bit signed mantissa and e_f is a signed 7-bit exponent. The format of the FLP is assumed to be normalized. The conversion consists of transforming the pair (m, e_f) to a single number e_x . The relationship between the two formats is given by

$$e_x = e_f + \log_2 m \quad (19)$$

where $\log_2 m < 0$, because m is normalized to lie in the range $1 < m < 5$.

The log function is implemented using a look-up table ROM of size $4K \times 12$ (48 Kbits). The ROM requires 12-bit addressing and produces a 12-bit output. The result of the conversion is calculated by adding the exponent to the output of the look-up ROM. The addition operation is an 18-bit add where the exponent is the MSBs of one input and the ROM output is the LSBs of the other input. The sign bits for both the mantissa and the exponent are passed through the conversion to produce a 20-bit logarithmic number of the following format.



A block diagram for the proposed conversion algorithm can be seen in Figure 4.

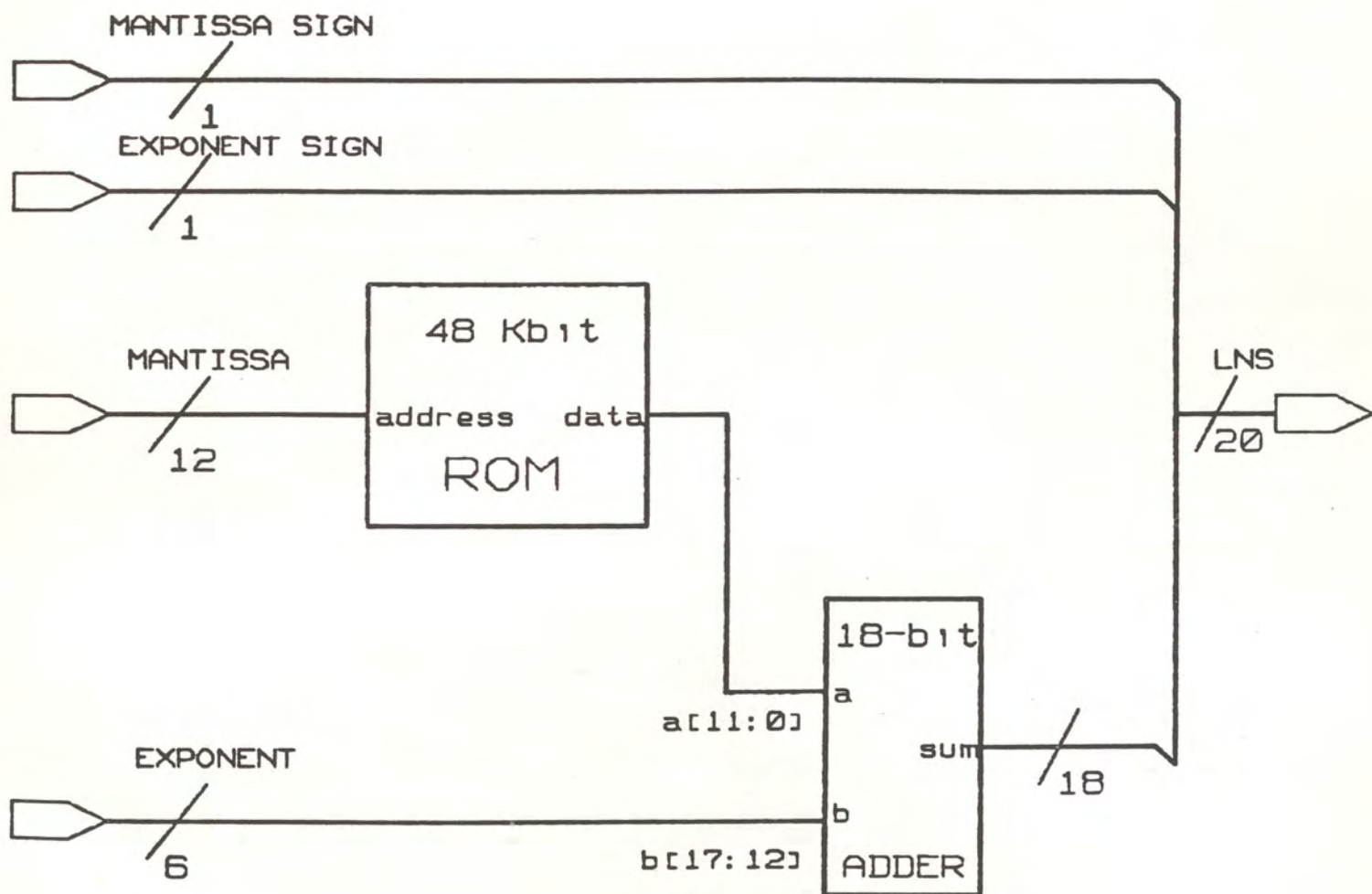


Figure 4. Floating Point to LNS Conversion Block Diagram.

LNS to Floating-Point Conversion

The LNS to FLP conversion algorithm consists of transforming a single number (e_x) into a pair of numbers (m, e_f), where

$$e_f = \text{ceiling}(e_x), \quad (19)$$

$$m = F(\text{ceiling}(e_x) - e_x), \quad (20)$$

and

$$F(k) = 2^k \quad (\text{or antilog}). \quad (21)$$

The computation of $\text{ceiling}(e_x) - e_x$ is nothing more than $1.0 - e_{xF}$, where e_{xF} is the fractional part of e_x . The calculation of $(1.0 - e_{xF})$ is the same as computing the two's complement of e_{xF} , ie $\bar{e}_{xF} + 1$. This computation can be avoided by programming a look-up table ROM in such a way that the ROM contains $F(x + 1)$ rather than $F(x)$ (Taylor et al. 1987).

The LNS to FLP conversions calculation of the exponent is done by finding the nearest integer to e_x which can be computed using an incrementor. The mantissa calculation requires a table look-up using the $F(x + 1)$ ROM. A block diagram for the described LNS to FLP conversion algorithm can be seen in Figure 5.

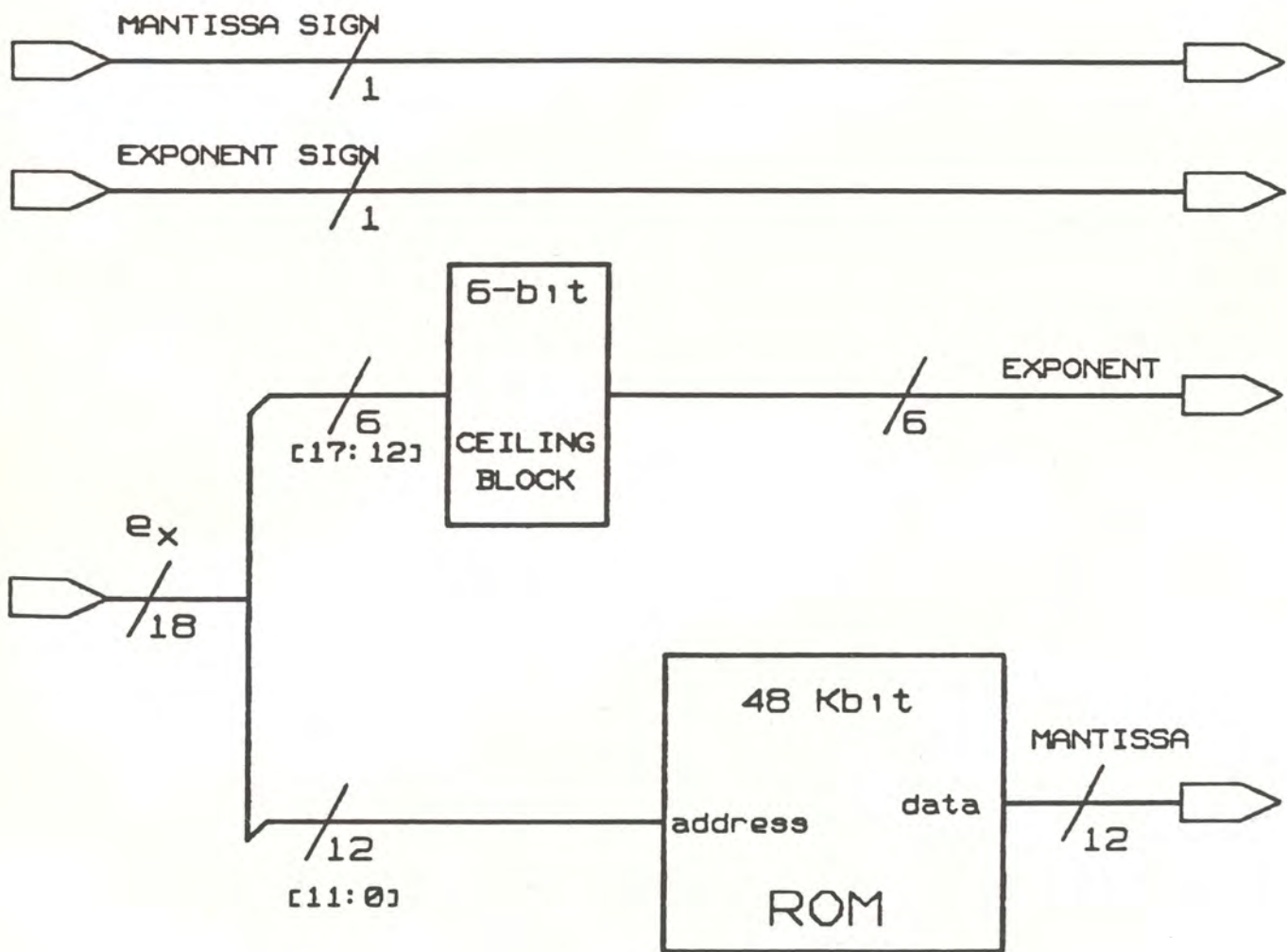


Figure 5. LNS to FLP Conversion Algorithm Block Diagram.

Conversion Chip

The cost of building a VLSI chip depends on the number of chips produced. A mask of the chip is generated from which any number of chips can be copied. The cost of a chip mask is very expensive, typically around \$50,000. The cost of using the mask to make a chip is relatively small, so the more chips produced the lower the unit cost.

Since it would be quite expensive to build masks for both a FLP to LNS conversion chip and a LNS to FLP conversion chip only one chip will be designed which contains both algorithms. This is possible due to the use of the 2 micron VLSI technology. The desired chip will require 96 Kbits of ROM, a 6-bit incrementor, an 18 bit fast adder, and some related control logic. The VTItools system was used to generate a conversion chip of size 178 x 156 mils which is well within the limits of current technology. A block diagram of the chip layout with relative component sizes can be seen in Figure 6. The CAD system was also used to compute time estimates for the two conversions which can be seen in Table 4. It should be noted that the conversion chip will be run at the same clock rate as the processing elements in the systolic array. The conversion chip will simple have registers on its inputs and outputs which are clocked at the system clock rate. This timing scheme will allow the conversion chip to be added to the systolic array without creating a noticeable timing difference in computational times due to its pipeline structure.

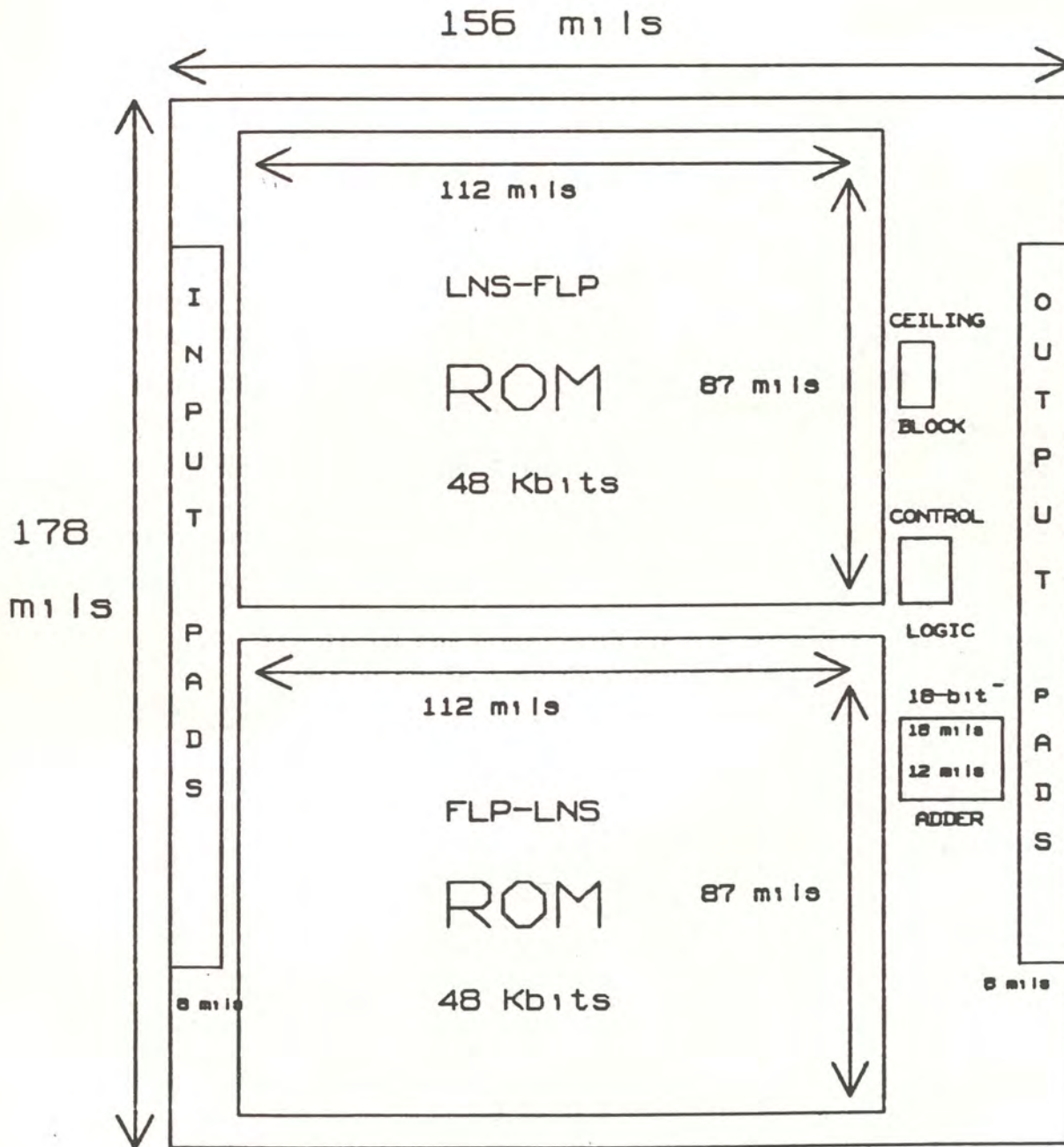


Figure 6. Block Diagram of Conversion Chip Layout.

TABLE 4
CONVERSION CHIP SPEED ESTIMATE

CONVERSION	OPERATION	TIME
FLP->LNS:		58ns
	18-bit add	20ns
	ROM look-up	38ns
LNS->FLP:		38ns
	6-bit increment	11ns
	ROM look-up	38ns

CHAPTER III, PROCESSING ELEMENT DESIGN

The major component of a systolic array is the Processing Element (PE). The purpose of this chapter is to investigate the design a high speed PE for use in the systolic array implementation of a Kalman filter.

The desired PE must use LNS to allow high speed arithmetical operations. The cell should also be capable of switching data to other cells will the ALU performs computations. The cell must contain the necessary hardware in the form of memory and a controller to allow programmability and reconfigurability. The PE should also be designed to take full advantage of parallel communications, both internally and externally. The proposed PE will contain a LNS ALU, ROM look-up tables, on board RAM, a control unit, internal and external data routing switches, and multiple internal and external buses. A block diagram of the proposed PE is shown in Figure 7.

The LNS ALU

A major component of the proposed PE will be the LNS based ALU. The ALU provides the high speed arithmetic calculations needed to implement the Kalman filter. The unit will compute six arithmetic functions, consisting of $X+Y$, $X-Y$, $X*Y$, X/Y , X^2 , and \sqrt{X} . Memory reduction techniques will be utilized to greatly reduce the amount of look-up table memory that is necessary to construct a LNS based ALU. The following sections will give a general overview of the LNS ALU (Taylor et al. 1987).

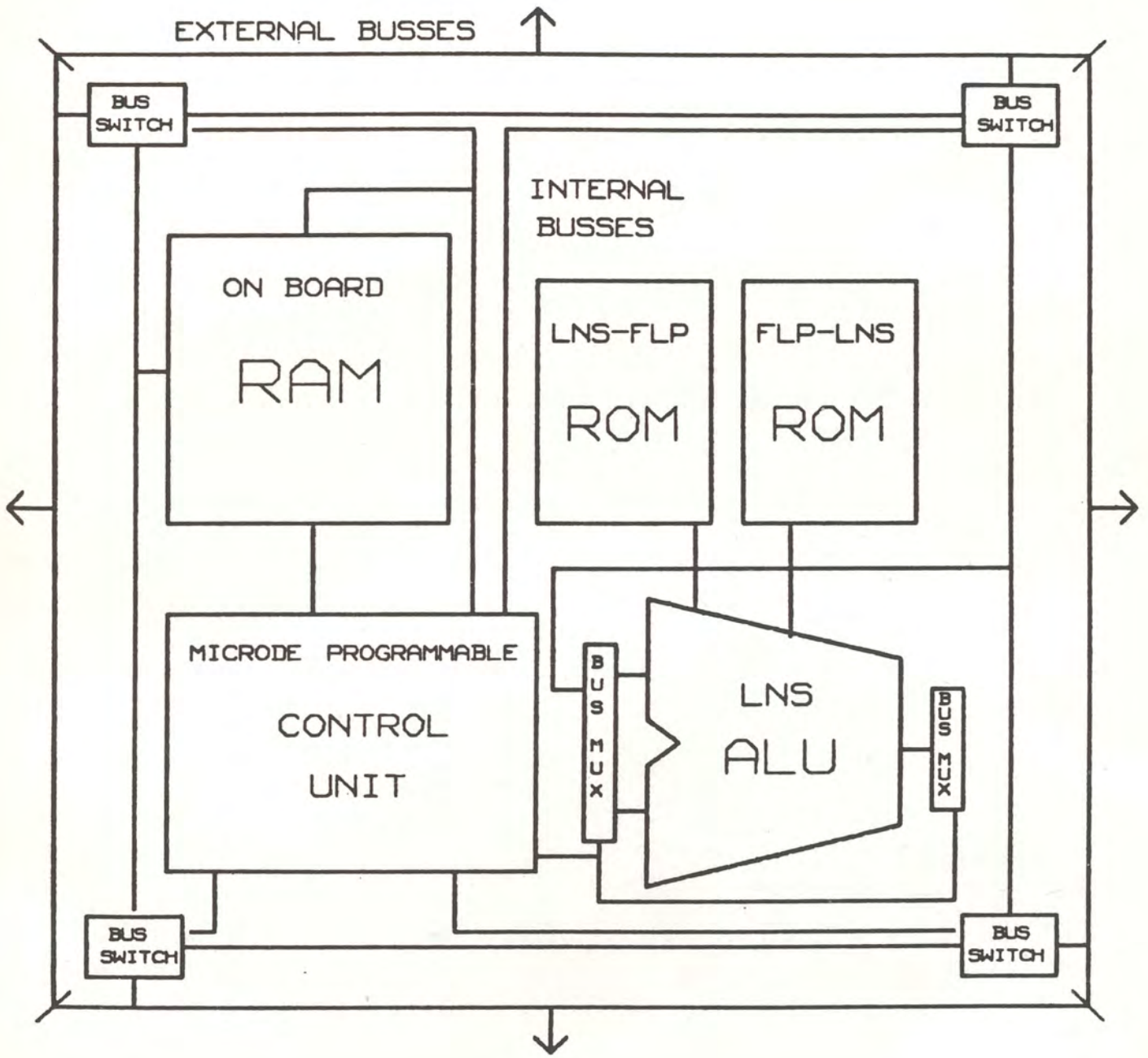


Figure 7. Block Diagram of Proposed PE.

Description of Arithmetic Algorithms

The numbers in LNS are represented as $X = (-1)^{S_{rx}} \cdot r^{e_x}$ where $r = 2$ and e_x is a 19-bit two's complement number with a 6-bit integer part e_{xI} , a 12-bit fraction part e_{xF} , and a sign bit S_{e_x} . S_{rx} represents the sign of the radix, and S_{e_x} the sign of the exponent. The LNS numbers are represented in the following word format.

S_{rx}	S_{e_x}	18-bit exponent magnitude
----------	-----------	---------------------------

In describing the arithmetic algorithms for the ALU e_x and e_y will be used to represent the numbers $X = 2^{e_x}$ and $Y = 2^{e_y}$ respectively. The result of the arithmetic operation will be e_z which represents the number $Z = 2^{e_z}$.

An important operation in the ALU is the multiplication function. The result of a multiplication is simply the addition of the exponents. The multiplication of two numbers X and Y may be written as

$$Z = X*Y = (-1)^{S_{rx} + S_{ry}} \cdot r^{e_x + e_y} \quad (22)$$

where

$$S_{rz} = S_{rx} + S_{ry} \quad \text{and} \quad e_z = e_x + e_y. \quad (23)$$

The next function to be discussed is the division operation. The division algorithm is almost identical to the multiplication algorithm since the function can be computed by subtracting the exponents. The division of two numbers can be expressed as

$$Z = X * Y = (-1)^{S_{rx} - S_{ry}} * r^{e_x - e_y} \quad (24)$$

where

$$S_{rz} = S_{rx} + S_{ry} \quad \text{and} \quad e_z = e_x - e_y. \quad (25)$$

The addition function is not as easy to implement as was multiplication or division. The addition operation requires a memory look-up to determine the LOG_2 of an input. The sum of two numbers X and Y is defined as

$$Z = X + Y = (-1)^{S_{rx}} * r^{e_x} + (-1)^{S_{ry}} * r^{e_y} \quad (26)$$

and

$$Z = X + Y = (-1)^{S_{rx}} * r^{e_x} (1 + (-1)^{S_{ry} - S_{rx}} * r^{e_y - e_x}) \quad (27)$$

where $S_{rz} = F1(e_x, e_y, S_{rx}, S_{ry})$ and the function $F1$ is defined in Table 5. The resulting sum is represented as

$$e_z = e_x + LOG_2(1 + r^{e_y - e_x}). \quad (28)$$

However, to reduce memory requirements e_z is actually implemented as

$$e_z = e_{\max} + LOG_2(1 + r^{e_{\min} - e_{\max}}) \quad (29)$$

where $e_{\max} = \max(e_x, e_y)$ and $e_{\min} = \min(e_x, e_y)$.

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

$$Z = X - Y = (-1)^{S_{rx}} * r^{e_x} - (-1)^{S_{ry}} * r^{e_y} \quad (30)$$

and

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

where $S_{rz} = F2(e_x, e_y, S_{rx}, S_{ry})$ and the function $F2$ is defined in Table 5. The resulting difference is represented as

$$e_z = e_x + LOG_2(1 - r^{e_y - e_x}). \quad (32)$$

Here again, as in addition, e_z is represent as

$$e_z = e_{\max} + LOG_2(1 - r^{e_{\min} - e_{\max}}) \quad (33)$$

to reduce the memory requirements.

The look-up tables used in the addition and subtraction algorithms can be combined into one table by the following definitions. First a term D is introduced which is defined as

$$-D = e_{\min} - e_{\max}. \quad (34)$$

The addition and subtraction functions can now be shown to involve either of two operations, $LOG_r(1 + r^{-D})$ or $LOG_r(1 - r^{-D})$. These two functions are implemented using a table look-up ROM, with D as the address input. The ROM actually holds two tables, one for the $F1$ function and one for the $F2$ function. The two tables can be referred to as $LU = 0$ and $LU = 1$ for addition (function $F1$) and subtraction (function $F2$) respectively. The logic for the addition and subtraction functions can be seen in Table 5.

TABLE 5
LOGIC FOR THE ADDITION AND SUBTRACTION FUNCTIONS

OPERATION	S_{rx}	S_{ry}	$e_x > e_y$	S_{rz}	LU	OUTPUT
add	0	0	yes	0	0	$e_x + \text{ROM out}$
add	0	0	no	0	0	$e_y + \text{ROM out}$
subtract	0	1	yes	0	1	$e_x + \text{ROM out}$
subtract	0	1	no	1	1	$e_y + \text{ROM out}$
subtract	1	0	yes	1	1	$e_x + \text{ROM out}$
subtract	1	0	no	0	1	$e_y + \text{ROM out}$
add	1	1	yes	1	0	$e_x + \text{ROM out}$
add	1	1	no	1	0	$e_y + \text{ROM out}$

The remaining two functions that the ALU can perform are the square and square root operations. These functions are fairly trivial to implement requiring only a shift left or shift right by one bit. The square of the number X can be expressed as

$$e_z = r^{e_x} < < 1 \text{ bit.} \quad (34)$$

The square root of the number X is represented as

$$e_z = r^{e_x} > > 1 \text{ bit.} \quad (35)$$

The square and square root operations are simply to implement in the LNS ALU, however, it should be noted that a larger than 20 bit register should exist so that accuracy is not lost when squaring or taking the square root of numbers.

Memory Reduction Technique

The addition and subtraction operations in LNS require a look-up table to evaluate the two functions $LOG_2(1 + 2^{-D})$ and $LOG_2(1 - 2^{-D})$. If the 19-bit address D was used in a "brute force" memory scheme to produce a 19-bit output the memory requirements would be extremely high (.5 Mbits x 19). A memory of this size would be difficult to build even using current VLSI technology. However, a preliminary paper by Taylor et al. presented a number of techniques that can be employed to reduce the memory to a practical size (Taylor et al 1987).

The first technique is to ensure that D is always positive by comparing e_x with e_y . In other words $D = \max(e_x, e_y) - \min(e_x, e_y)$. In the actual hardware a comparator is not used, instead $e_x - e_y$ and $e_y - e_x$ are evaluated in parallel and the appropriate one chosen to save time. The result of ensuring that $D > 0$ is that $F1$ is always less than one and therefore only needs a 12-bit wide output instead of a 19-bit wide output.

Taylor et al also determined that for some values of D the result of $F1(D)$ and $F2(D)$ would be smaller than what could be represented in a 20-bit ALU. The value of D determined to produce this case is 12.5 for a 20-bit system. Therefore, for all values of $D > 12.5$ the value is known to be zero and need not be looked up. Another advantage is that to represent numbers less than 12.5, the address input D need only have four bits to the left of the fraction point. This produces a 16-bit address (4-bit integer, 12-bit fraction) instead of a 19-bit address. This method alone will reduce the memory requirements by a factor of eight (Taylor et al. 1987).

A third technique is to break up the address space in such a way so as to have smaller sized memory spaces rather than one or two large ones. The graphs of the LOG_2 functions are ones that become extremely flat, almost a straight line, for large values of the variable D . Therefore, it makes sense to partition D -space into smaller parts so that intervals of D that are more to the right (i.e. large D) use smaller ROMs than intervals to the left. Using this method various points along the D axis have entirely different mapping ratios. The mapping ratio around a point is defined as the density of points on the D axis divided by the density of points on the $F1(D)$ or $F2(D)$ axis. Points are then chosen on the D axis such that the mapping ratio is close to some power of two. This will allow the elimination of $LOG_2(\text{mapping ratio})$ number of bits from the address width of D . The two functions $F1(D)$ and $F2(D)$ will have the same mapping ratios and thus the same address width.

The ROM size can be reduced even further since the $F1(D)$ and $F2(D)$ functions monotonically decrease. The most significant bits of the 12 bits can be reduced in certain ranges of D . The ranges of D along with the needed ROM size was determined using the techniques presented by Taylor et al. and can be seen in Table 6. The range of D was broken up into 10 ROMs with values of $D > 9.0$ implemented in random logic.

The memory requirements for each LOG_2 function was determined to be 77 Kbits using memory reduction techniques that were discussed previously. The total memory requirements for both functions $F1$ and $F2$ are 154 Kbits, which is well within the size

TABLE 6
MEMORY REQUIREMENTS FOR ONE LOG_2 FUNCTION

ROM #	D RANGE	$F(D)$ WIDTH	ROM SIZE	TOTAL BIT SIZE
1	0.0-0.5	11	2K x 11	22 Kbits
2	0.5-1.0	11	1K x 11	11 Kbits
3	1.0-2.0	12	2K x 12	24 Kbits
4	2.0-3.0	11	1K x 11	11 Kbits
5	3.0-4.0	10	512 x 10	5 Kbits
6	4.0-5.0	9	256 x 9	2.3 Kbits
7	5.0-6.0	8	128 x 8	1 Kbits
8	6.0-7.0	7	64 x 7	.45 Kbits
9	7.0-8.0	6	32 x 6	.2 Kbits
10	8.0-9.0	5	16 x 5	.08 Kbits

that can be constructed using VLSI technology. The memory reductions techniques will thus allow the construction of a 20-bit LNS ALU for use in the systolic array implementation of the Kalman filter.

ALU Components

The basic LNS processor is an asynchronous device that will perform six operations: multiplication, division, addition, subtraction, square, and square root. The ALU accepts a 3-bit op-code and two 20-bit LNS operands, e_x and e_y . The single operand operations are performed on e_x only. The processors output is a 21-bit LNS number and a system overflow flag. The main components of the ALU are four 19-bit two's complement adders/subtractors, a 20-bit shift register, and look-up table memory implemented in ROM.

The 19-bit adders are all implemented as carry look-ahead adders to allow for minimum computation times. The use of look-ahead carry adders requires some extra room on the VLSI chip, but produces a significant computational time savings to be advantages. The adders will accept two 19-bit two's complement operands and produce the two's complement sum or difference as output. In order to save space two of the adders which compute difference only are modeled as pure subtractors which saves some control logic. The third and fourth adders are selectable between addition or subtraction, where the subtraction simply inverts the desired input and forces the carry in input to one.

The shift register accepts a 20-bit input and shifts it right or left by one bit depending on the control signal. If the register shifts right it fills with the sign bit, while a shift left fills with zero. The shift register is used to perform the square and square root computations.

The look-up table memory is actually composed of two sets of ten ROMs and some random logic. The two sets of look-up tables are used to obtain the values of the two functions $LOG_2(1 + 2^{-D})$ and $LOG_2(1 - 2^{-D})$. The correct set of ROMs is determined by the select signal LU which specifies which LOG_2 function is being evaluated. The random logic determines which interval the input D belongs in and selects the appropriate ROM. The select lines to the ROMs are simply a function of the bits of the address D .

The ALU contains some other miscellaneous logic in terms of multiplexers to switch input data to the correct hardware and logic gates that are used to implement control signals for select lines and enables. Having defined the major components of the ALU, the next section will show the hardware used to implement the various arithmetic functions and control logic.

ALU Data Paths

Similarities were found in many of the arithmetic functions performed by the ALU. In an attempt to save hardware and reduce chip size, functions that were similar in nature were combined into single data paths. The result of this methodology was the creation of three separate data paths and some control logic. The first path implements the multiplication and division functions, the second path is used for addition and subtraction, and the third data path performs the square and square root functions. The data paths are also designed to allow parallel computation, otherwise more than one data path can be functional at any point in time.

The multiplication and division data path is quite simple in construction, using only a 19-bit adder/subtractor. The data path either adds or subtracts e_x and e_y depending on the op-code. A block diagram of the multiplication and division data path can be seen in Figure 8.

The addition and subtraction data path is rather complex requiring the use of look-up tables to compute a LOG_2 function. The path also contains three adders and a two-to-one mux. The first two adders are used to compute the two differences $e_x - e_y$

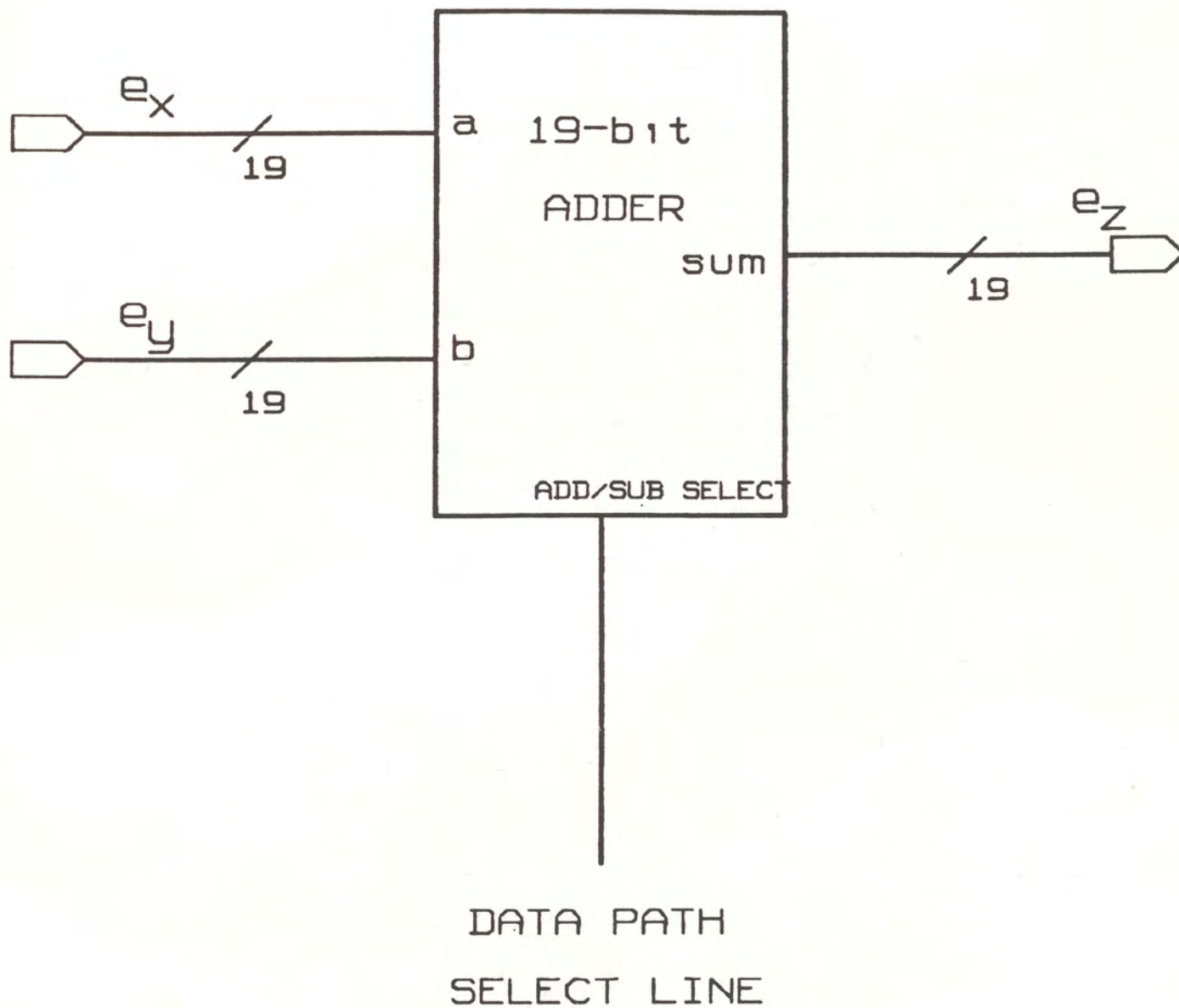


Figure 8. Multiplication/Division Data Path.

and $e_y - e_x$ in parallel. A two-to-one mux then selects the desired value depending on which result is negative. This allows the sign bits to be gated to form a select for the multiplexer. The third adder is used to add the look-up table value to e_x to produce the result. A block diagram of the addition and subtraction data path is shown in Figure 9.

The final data path contains the square and square root calculations. The only component in the data flow is a 20-bit shift register. The register simply shifts the e_x input right or left to implement the square or square root functions respectively. A block diagram of the square and square root data path is shown in Figure 10.

In order to implement the various arithmetic functions into a single unit it is necessary to construct some type of control logic. There are two major tasks that the control logic must perform: selecting the desired arithmetic function and handling an arithmetic overflow. The basic job of selecting which function to perform is handled by using three control lines, one to each data path. The signals are generated outside the ALU by the PE control unit. The control signals simply enable the different data paths. The overflow logic handles the case where the resulting number is larger than the system can output. In this case the logic simply outputs the largest representable number and sets the overflow flag.

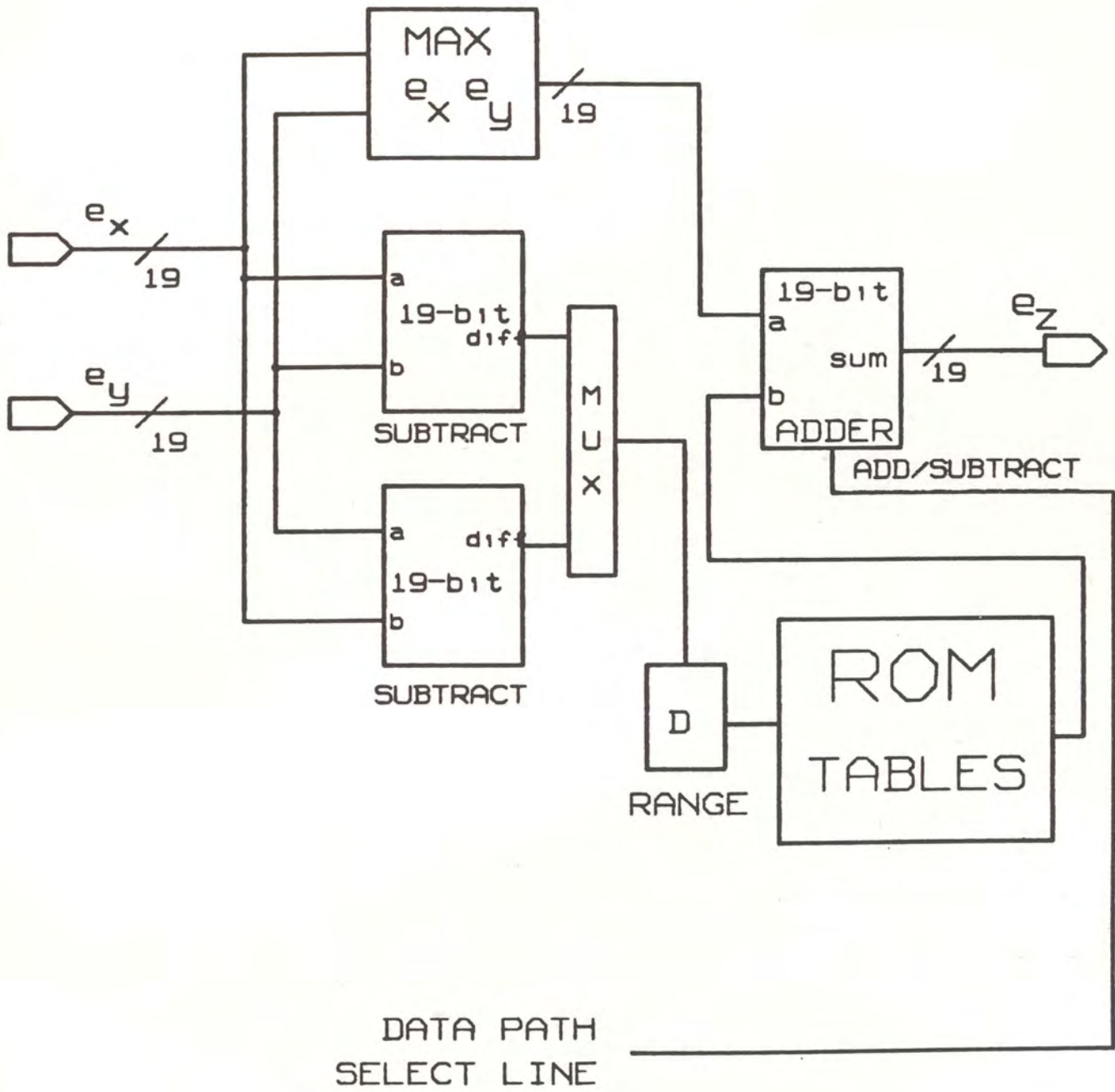


Figure 9. Addition/Subtraction Data Path.

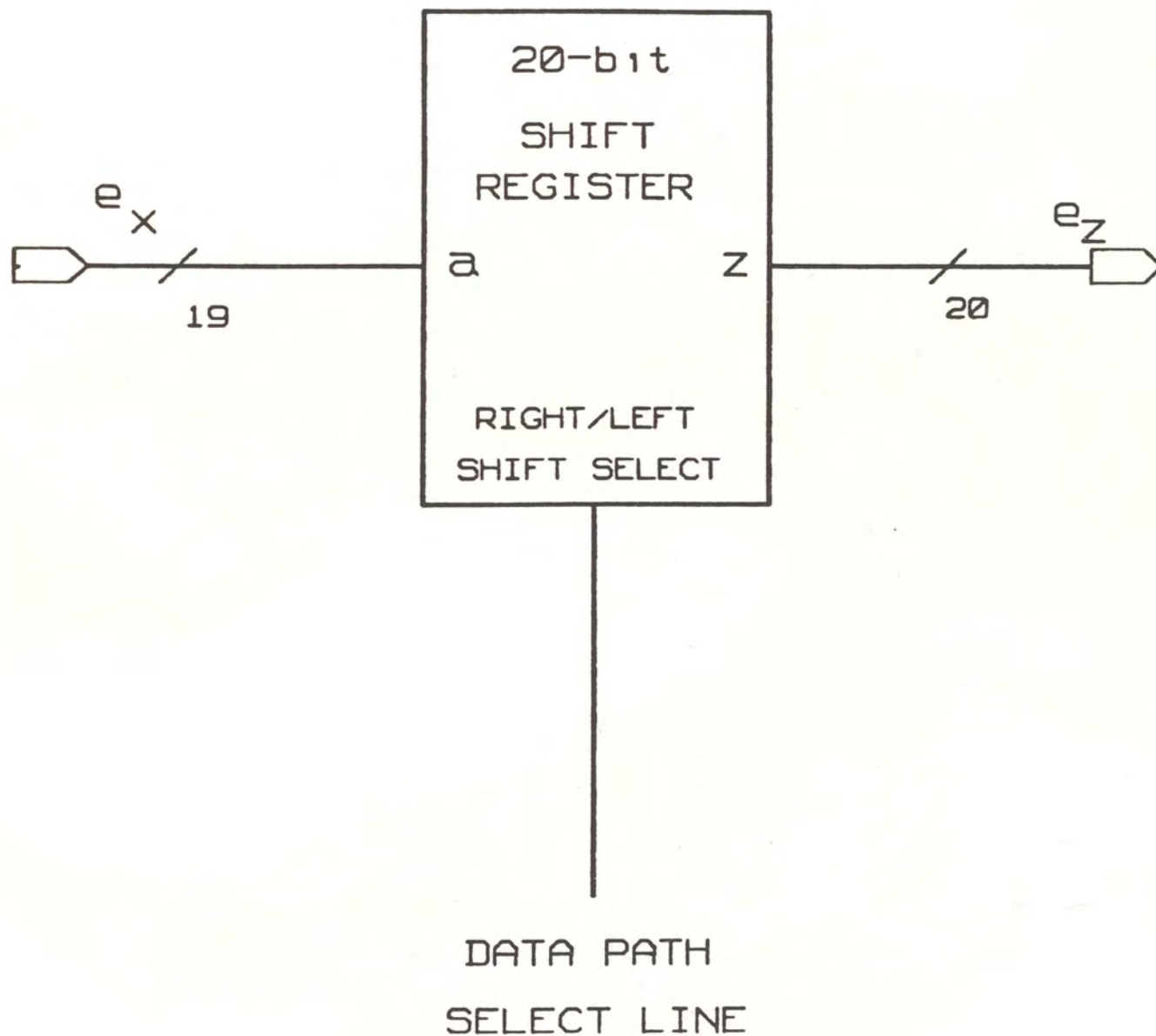


Figure 10. Square/Square Root Data Path.

There will also be some miscellaneous control logic necessary for determining which look-up table to address or which of the difference calculations in the addition/subtraction database to choose. This logic is easily implemented using simple logic gates. At this point it should be noted that this paper will not present the actual ALU schematics, but instead is written more to show the feasibility of this type of ALU.

The previously defined arithmetic data paths and control logic are used to construct a block diagram of the LNS ALU. The block diagram of the ALU is displayed in Figure 11.

ALU Layout and Timing

The VTItools CAD system was used to produce speed and size estimates for the LNS ALU. However, the ALU was not effectively simulated since only size and speed models were used for the look-up table ROMs.

The VTItools system was used to determine the time required to compute each of the arithmetic functions in the ALU. The decode and control logic was found to have an overhead on each function of approximately 10ns. The control logic used will be discussed in a later chapter and is only mentioned here for the calculation of ALU timing. The individual components in each functions data path were then analyzed to determine their delay. The delays were then totaled to arrive at a final value for each function. The results of the timing analysis are shown in Table 7.

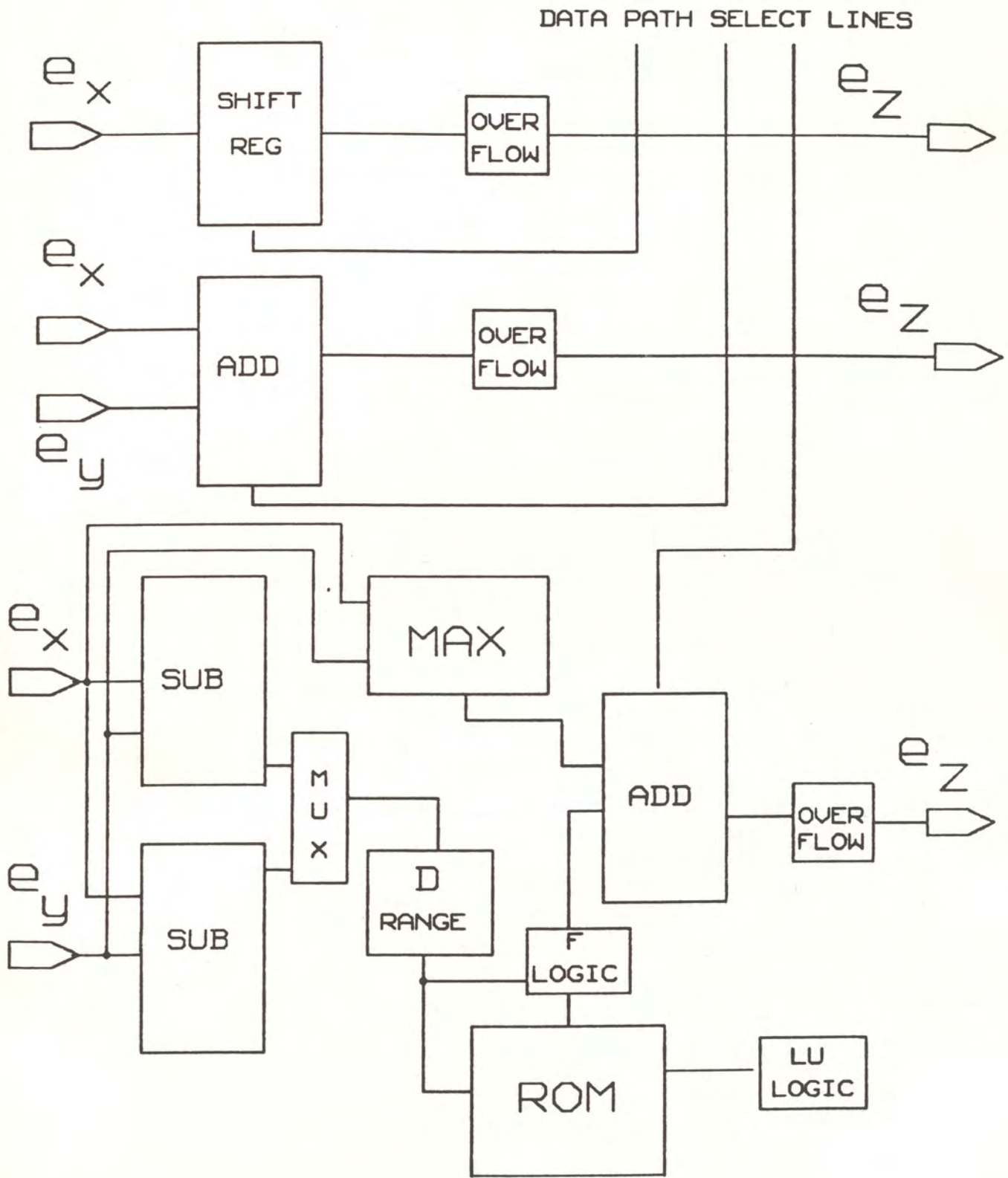


Figure 11. ALU Block Diagram.

TABLE 7
ALU COMPONENT AND FUNCTION TIMING

ITEM	FUNCTION					
	MULT	DIV	ADD	SUB	SQUARE	SQUARE ROOT
control	10ns	10ns	10ns	10ns	10ns	10ns
19-bit add	22ns	22ns	22ns	22ns	-	-
19_bit sub	-	-	22ns	22ns	-	-
20-bit shift	-	-	-	-	5ns	5ns
ROM table	-	-	28ns	28ns	-	-
total	32ns	32ns	82ns	82ns	15ns	15ns

The ALU size was computed by capturing the schematics of the major hardware components and performing a route. The VTI route command will produce a floorplan for the entered circuit with dimensions given in mils. A diagram of the generated floorplan along with the corresponding dimensions is shown in Figure 12.

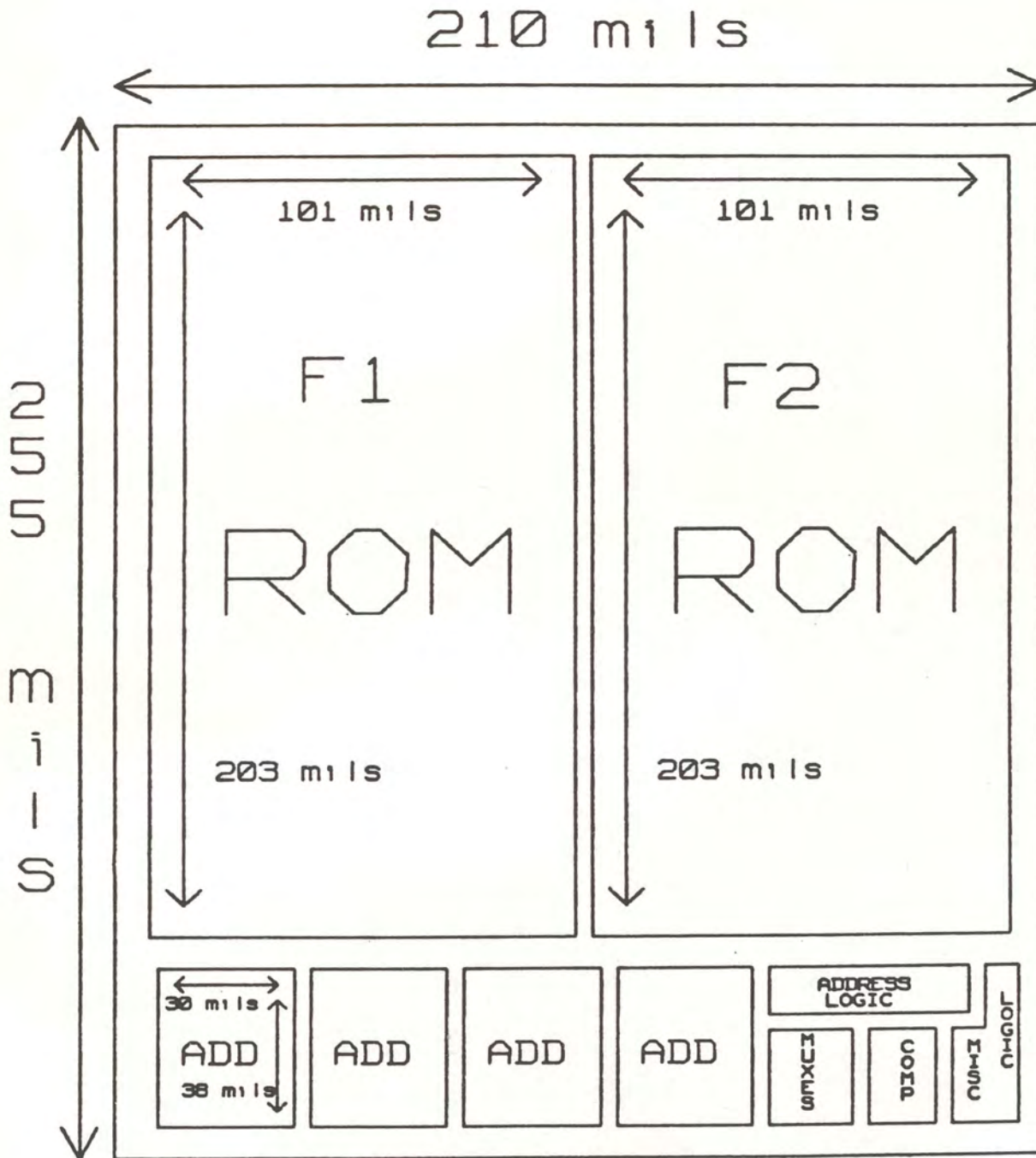


Figure 12. ALU Floorplan.

PE Interface Structure

An important part of designing a high speed PE is the creation of a highly parallel interface structure. The proposed PE will be used in an orthogonal array architecture and must be capable of parallel communication to each of its neighboring cells. The following sections will discuss a design which will support the required PE parallelism in the interface structure.

Internal Interfaces

To support the maximum amount of parallelism the proposed PE must be capable of routing data to three main areas, the ALU inputs, a data storage memory, or to an external bus. The PE will contain eleven internal buses which will allow communication to any of the ALU inputs or the storage memory. The data paths for each ALU function will have their own internal bus for both input and output. Two buses will be dedicated to routing data to and from the storage memory. Since the storage memory will be a dual port RAM one bus will be connected to each port. The remaining bus will be used as an ALU bypass bus to allow data to pass straight through the PE. The internal buses will all have a width of 20 bits to correspond to the previously designed LNS ALU.

The PE will use internal bus switches on each of the ALU outputs to allow data to be routed back to a desired ALU input, to the storage memory, or simply pass through to the one of the output buses. The bus switches will each be controlled by the use of a 3-bit op-codes which are shown in Table 8.

TABLE 8
OP-CODES FOR INTERNAL BUS SWITCHES

OP-CODE	FUNCTION
000	pass data through PE
001	route data to square/square root input
010	route data to e_x mult/div input
011	route data to e_y mult/div input
100	route data to e_x add/sub input
101	route data to e_y add/sub input
110	route data to storage memory port 1
111	route data to storage memory port 2

The storage memory will also contain two internal bus switches to allow data routing to any of the ALU input buses. The memory switches are also capable of connecting to the PE bypass bus. One memory switch is assigned to each port on the memory storage RAM. A 3-bit op-code is used to control each storage memory bus switch, were the values and corresponding functions can be seen in Table 9. It should be noted that the memory switch has an unused op-code which could be utilized in the future if modifications were made to the PE. The internal bus structure having been designed, it will next be necessary to interface the PE to the outside world.

TABLE 9
OP-CODES FOR MEMORY BUS SWITCHES

OP-CODE	FUNCTION
000	switch port to square/square root input
001	switch port to e_x mult/div input
010	switch port to e_y mult/div input
011	switch port to e_x add/sub input
100	switch port to e_y add/sub input
101	switch port to bypass bus
110	do not switch data to any bus
111	do not switch data to any bus

External Interface

The proposed PE is to be configured in a orthogonal array structure. In order to communicate to any of its neighboring cells in parallel the PE must contain four external data buses. The data buses will be controlled using external bus switches which will allow the data to be connected to many of the internal PE buses.

A separate bus switch will be allocated to each external bus. The switch is a bidirectional data switch that will simply connect the external bus to the desired internal bus. The external bus switches are controlled by a 4-bit op-code which is described in Table 10. The external bus switches each contain a latch to trap data. The data that is sent or received by the switch will be latched and thus present for one full clock cycle. It should be mentioned that at the present time there are four extra op-codes which could be used in the future if more external buses were added to

the PE. The external buses all have a width of 20 bits to agree with the already defined internal bus structure. A block the diagram of the combined internal and external bus organization is shown in Figure 13.

TABLE 10
OP-CODES FOR EXTERNAL BUS SWITCHES

OP-CODE	FUNCTION
0000	switch data to bypass bus
0001	switch to square/square root input
0010	switch to square/square root output
0011	switch to e_x mult/div input
0100	switch to e_y mult/div input
0101	switch to e_z mult/div output
0110	switch to e_x add/sub input
0111	switch to e_y add/sub input
1000	switch to e_z add/sub output
1001	switch to memory bus port 1
1010	switch to memory bus port 2
1011	no data enters PE
1100	no data enters PE
1101	no data enters PE
1110	no data enters PE
1111	no data enters PE

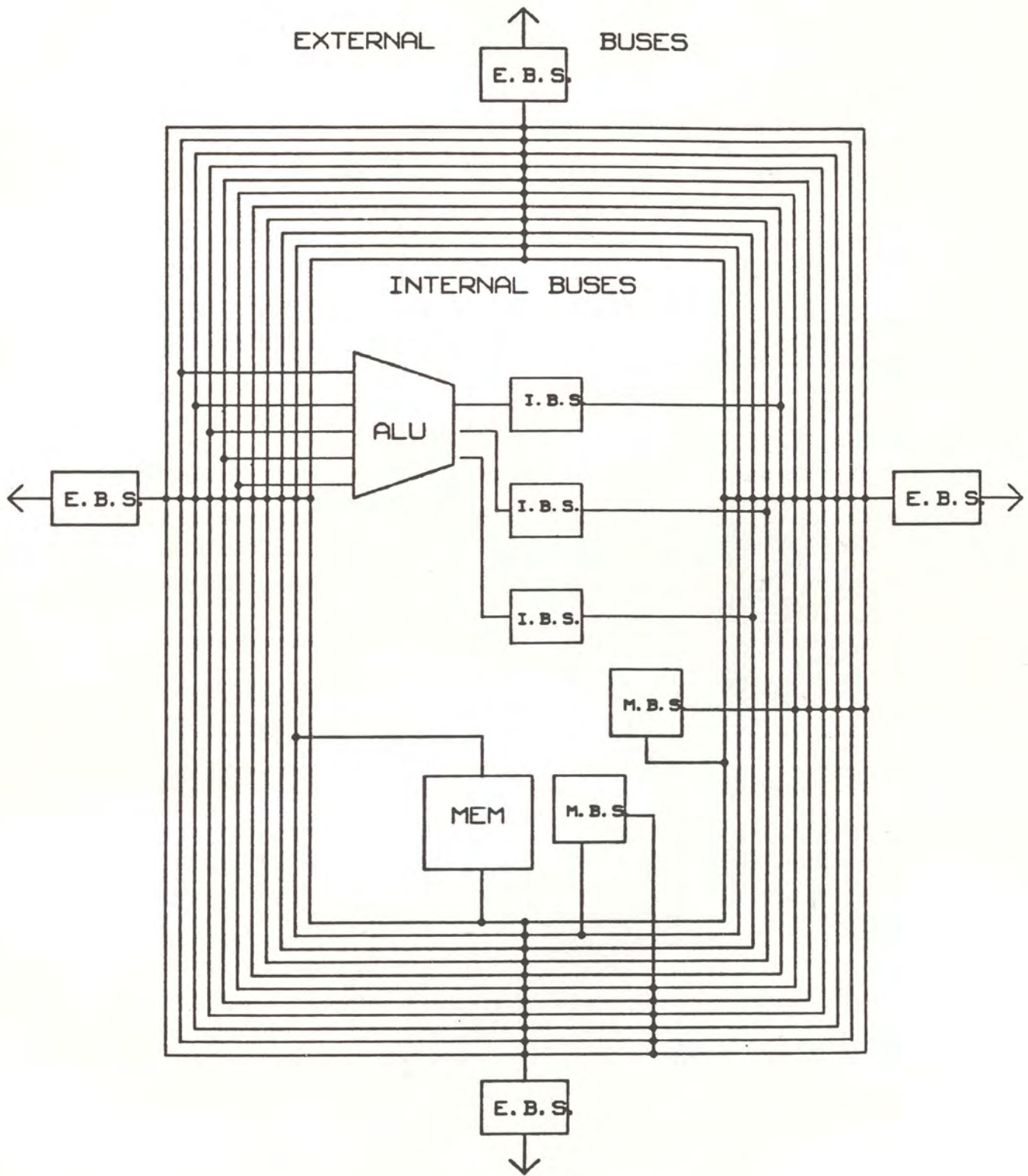


Figure 13. Block Diagram of PE Bus Structure.

PE Control Structure

Another important aspect of the proposed PE is the control structure. A well designed control structure will allow the PE to support a maximum amount of parallelism. The control structure will also define how generic the PE is, in other words, the number of different algorithms that can be implemented using the PE.

The control of the proposed PE will be performed by the use of a microcode control unit. The use of a microcode controller will allow the required flexibility to be designed into the control structure. The microcode unit will be simplified by using straight line code which does not require condition statements. The controller will simply be a 1K x 23 bit memory in which the control bits will be stored. The control structure will include four counter/comparator blocks which will allow the user to define the sections of memory that should be run in loops.

Counter/Comparator Block

The counter/comparator blocks will be used to define algorithms for the PE. The counter/comparator block is loaded with the starting and ending addresses which it will loop through in the microcode memory. The corresponding memory locations are then loaded with the necessary control bits which will perform the desired PE function. A control bit is used to enable the counter whenever a programmed algorithm is to be executed. The counter simply produces the read address for the microcode memory. The resulting memory output data is then used to determine the PE function. The counter will continue to increment the read address until the ending address loaded

into the comparator is reached. The counter is then reset to the starting address beginning the loop over.

The reading of data from the microcode memory will take approximately 38 ns, which would be a rather large overhead for all the PE functions. In order to reduce this overhead a negative clock will be used to clock the counters and microcode memory. The read operation occurs asynchronously so the delay due to the counter will not affect the memory operation. The negative clocking of the counter will allow the control data to be used to set up the PE before the system clock occurs. The system clock is used to clock the registers in the PE and the data storage memory. It should be noted that the negative clock will be generated internally to the PE by using the entered system clock. The use of these counter/comparator block scheme will allow the PE to be configured for a variety of algorithms and greatly reduce the amount of external PE control.

Control Words

The control of the PE will require the use of two consecutive words in the microcode memory. The first word will be used to control the external bus switching and the ALU. The second word will control internal bus switching and the storage memory.

The first control word will contain the 4-bit op-code for the external bus switches along with the read/write enable for the storage memory, the 3-bit memory address, and the 3-bit op-code for the ALU. The bus switch will use its 4-bit op-code to

Memory Organization

In order to produce the required control bits the microcode controller needs two consecutive 23-bit memory words. If the system has to perform two consecutive memory reads there will be a memory access time worth of delay before the PE can perform an operation. In order to alleviate any delays the microcode memory will be organized as two 512 x 23 bit word memories. The read address generated by the counter/comparator will be sent to both memories so that both control words will be available at the same time. A memory scheme of this nature will ensure an efficiently operating PE. However, it may be necessary to write to each memory independently in certain situations. In order to access the microcode memories independently, separate enable bits will be connected to each memory. The enable bits will both be set to allow the memories to be accessed during the running of microcode.

The PE will also contain a dual port RAM to be used as a data storage memory. The RAM will be organized as a 16 x 20 bit memory. The first eight words will be used for the temporary storage of data. The remaining words can be used to implement a PE self test which will be described in a later section. The dual port RAM will allow the PE to perform both a read and a write operation in one system clock cycle. The storage memory will be connected to the rest of the PE with two buses, one bus being dedicated to each RAM port. Using the described memory organization and control word scheme a block diagram of the PE control structure was generated and is displayed in Figure 14.

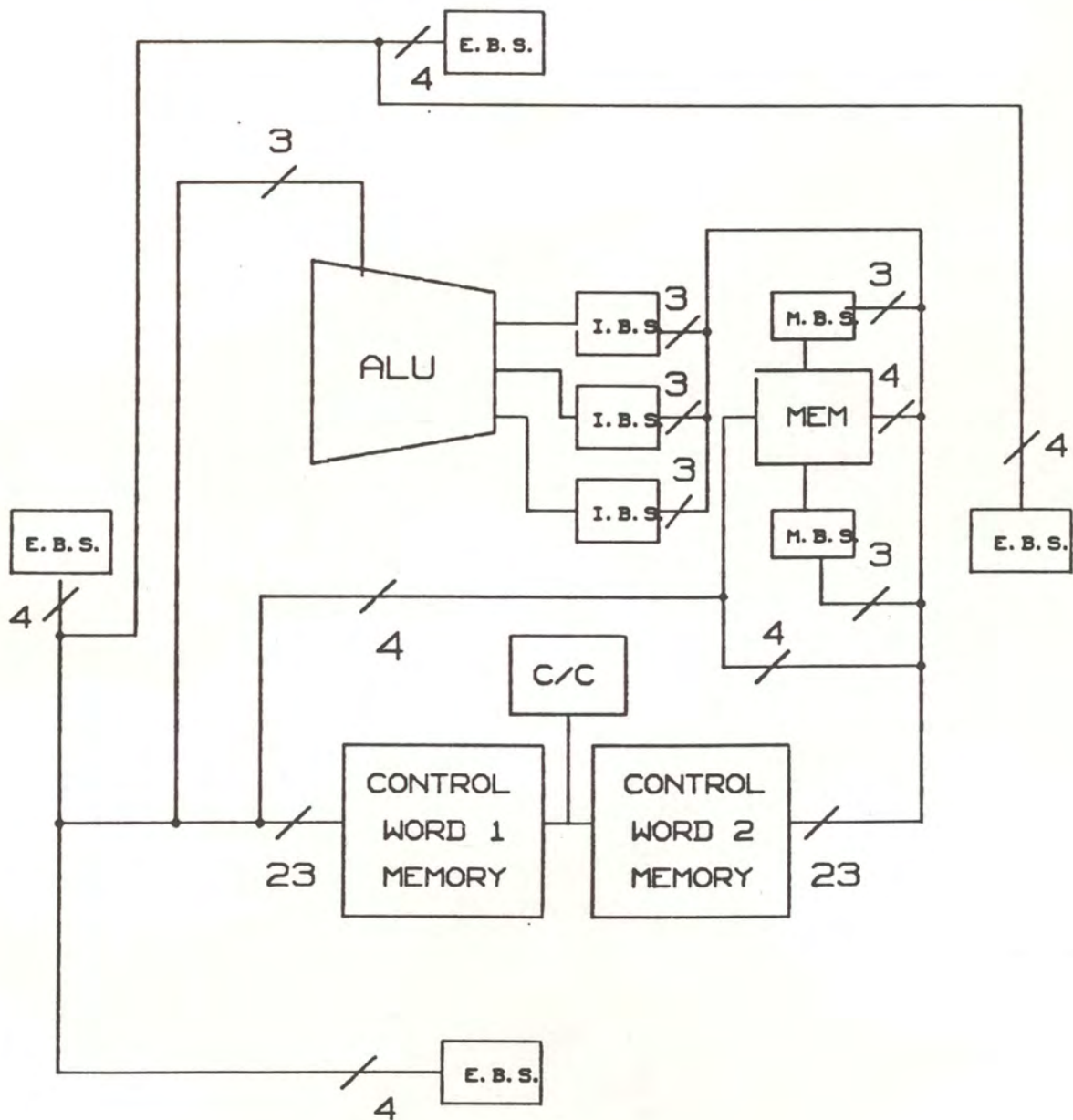


Figure 14. Block Diagram of PE Control Structure.

External Control

The PE will be externally controlled by a master controller in the systolic array. The controller will require the ability to select which counter/comparator to run, to set up start and end addresses for the microcode, and to program the microcode controller. A proposed control scheme that will meet these requirements will contain four control bits and the two memory enable bits. The functions performed by the four control bits are described in Table 11.

TABLE 11
OP-CODES FOR PE CONTROL

OP-CODE	FUNCTION
0000	null process (self test)
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 memory enable bits are used to determine which memory the user wishes to access during the reading or writing of microcode to memory.

Self Test

One of the operations the PE is capable of performing is the null operation, where the PE simply sits idle. However, it would be a waste of processor time to do nothing during the null mode. A possible function that could be performed during this available time is a PE self test. The extra eight words in the storage memory could be used to store data that would be entered into the ALU. The results of the ALU would then be compared to the known results also stored in memory. If the two values did not agree an error flag would be set. The addition of this test to the PE would require a 20-bit comparator and some random logic. The addition of this extra hardware is worth the assurance it will give that the PE is functional.

The PE will output two status flags to inform the master controller of its status. The previously described self test will produce an error flag that will indicate whether the PE is operational. It should be noted that the self test only checks the operation of the ALU and some internal buses. The PE will also output an ALU overflow flag that will indicate whether an arithmetic overflow has occurred. The master controller will be able to use these status flags to determine the accuracy of implemented algorithms in the systolic array.

Programming the PE Microcode Controller

The PE microcode controller is simply a pair of memories that store control bits. The memories can be written to by setting the correct op-code defined in Table 11 and enabling the desired memory. The data that the user wishes to store in memory is

entered on the external bus that would normally connect to the PE's neighboring left cell. In this way the PEs in the systolic array can be programmed a column at a time. The user may find it useful to write a basic compiler that would take defined words and translate them into the corresponding control bits that would implement the desired function. The user could then have PE functions such as multiplication, addition, and square root already defined so that a user not familiar with the PE design could still write microcode for the processor. This may be an interesting topic for later research, but will not be discussed in the scope of this paper.

System Clock

The final item to consider in the design of the PE control structure is the matter of system timing. The PEs in the systolic array should all be clocked simultaneously by a system clock. The LNS conversion chip will also be clocked by the same system clock to create a totally synchronize systolic array. The use of a system clock simply means that the PE and conversion chip will contain registers on both their inputs and outputs which will be clocked at the system rate. In the design of this PE a system clock of 100 ns was assumed to allow time for the worst case ALU functions of addition and subtraction (82ns) along with a system overhead margin. The major components of the PE have now been defined, the final task will be to integrate them into a working VLSI chip.

PE Simulation

The design of the PE is now complete, however, the PE must be tested to determine its effectiveness in implementing the Kalman filter equations. The PE will be tested by implementing in microcode two functions that are used repeatedly in the Kalman filter equations. The first function is

$$Z_{out} = A * B_{in} + Z_{in}, \quad (36)$$

and

$$B_{out} = B_{in} \quad (37)$$

where A is assumed to already be present in the PE. The values of Z_{in} and B_{in} are present on two of the PE's external buses. The second operation which will be implemented is

$$A_{out} = A_1, \quad (38)$$

and

$$A_1 = A_{in} \quad (39)$$

where A_1 is stored in the PE memory and A_{in} is present on one of the external buses. The control words necessary to perform these operations will be shown, along with the number of system clock cycles that are required to compute the function. It is assumed that the control words are stored in the microcode memory and that a counter/comparator is assigned to each function.

The control words specified will be of the format described earlier. The external bus switches are numbered one through four starting on the top of the PE and moving clockwise around the four sides. The internal bus switches are numbered one through three, where the first is placed on the multiple/divide output bus, the second on the add/subtract output bus, and the third on the square/square root output bus. The memory switches are assigned to port 1 and port 2 of the storage memory respectively. The op-codes that will be used in the control words were previously defined in tables 8 through 10.

The first function, described in equations (36) and (37) will take two system clocks to perform. The microcode for control words one and two will be displayed. During the first clock B_{in} will be placed on the e_x input of the multiple/divide path and simultaneously shifted out of the PE. The A value will also be read from memory and placed on the e_y input of the multiple/divide path while the Z_{in} value is stored in memory. The control words read from the microcode memory during the first clock should be as follows:

CONTROL WORD 1					
SWITCH 1	SWITCH 2	SWITCH 3	SWITCH 4	MEMORY	ALU
0011	1111	0011	1001	1001	000

CONTROL WORD 2						
SWITCH 1	SWITCH 2	SWITCH 3	MEMORY			
000	000	000	111	010	0000	0000

The second clock will allow Z_{in} to be added to the result of $A*B_{in}$ and shifted out of the PE on an external bus. The control words that would be placed in the microcode memory are as follows:

CONTROL WORD 1					
SWITCH 1	SWITCH 2	SWITCH 3	SWITCH 4	MEMORY	ALU
1111	1000	1111	1111	0000	000

CONTROL WORD 2						
SWITCH 1	SWITCH 2	SWITCH 3	MEMORY			
100	000	000	100	111	0001	0000

A flow diagram showing the implementation of the first functions microcode in the PE is displayed in Figure 15. The data flows internal and external to the PE are shown along with the corresponding clock times.

The second function which is described in equations (38) and (39) will also take two system clock cycles to execute. The first clock will allow A_1 to be shifted out of the PE and A_{in} to be stored in the storage memory. The microcode used to perform these functions is shown below.

CONTROL WORD 1					
SWITCH 1	SWITCH 2	SWITCH 3	SWITCH 4	MEMORY	ALU
1001	1111	0000	1111	0011	000

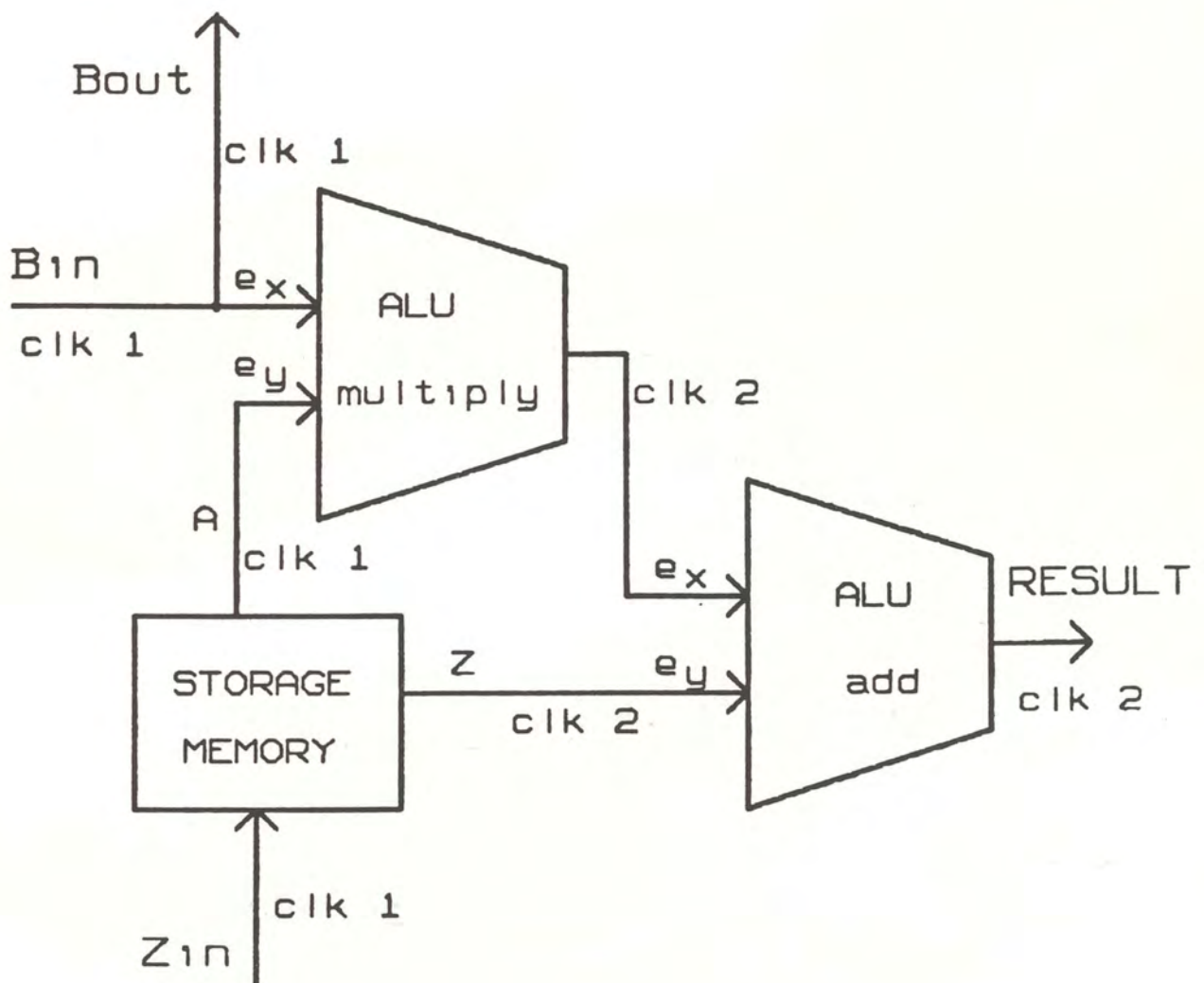


Figure 15. Flow Diagram of First Microcode Function.

CONTROL WORD 2						
SWITCH 1	SWITCH 2	SWITCH 3	MEMORY			
000	000	000	111	101	0000	1100

In the first clock cycle A_1 was read from memory location 3 while A_{in} was stored in memory location 4. This is due to the fact that a read and write operation cannot be made in the same clock cycle to the same address using the defined control structure. In order to allow this code to be repeatable the value of A_{in} must be moved from memory location 4 to memory location 3. The second clock cycle uses the following control words to perform the required memory move operation.

CONTROL WORD 1					
SWITCH 1	SWITCH 2	SWITCH 3	SWITCH 4	MEMORY	ALU
1111	1111	1111	1111	0000	000

CONTROL WORD 2						
SWITCH 1	SWITCH 2	SWITCH 3	MEMORY			
000	000	000	101	101	0100	1011

A flow diagram which describes data paths and clock cycles for the second function is shown in Figure 16.

The above examples of microcode demonstrate the ability of the PE to implement functions necessary for the Kalman filter equations. The microcode was assumed to already be present in memory since the task of loading microcode is more a function of the systolic array controller. The user would then simply enable the correct counter

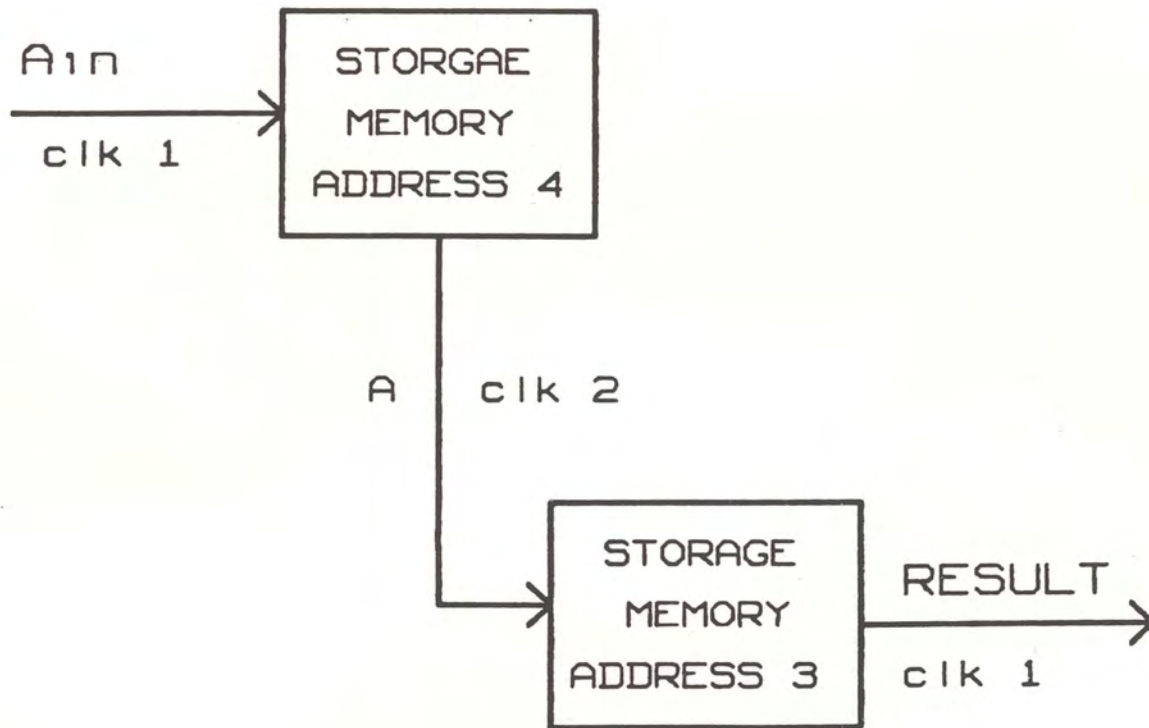


Figure 16. Flow Diagram for Second Microcode Function.

which is assigned to the desired function. It should be noted that the microcode used to implement the desired functions is not the only microcode implementation that would work. It is possible to use different microcode to perform the same functions. The examples were given to show the ability of the PE to perform the operations, and not to illustrate the absolute method of implementing the function.

Chip Layout and Key Parameters

The final chip layout was performed using the VTItools CAD system. The first step was to determine the total I/O to the PE. The next step was to place the PE's major components such as the ALU and memories in a position that would minimize internal routing. After the PE had been routed an accurate measurement of chip size and speed could be calculated using the CAD system.

Chip I/O

The PE chip was found to have a total of 89 I/O paths. The majority of the I/O is allocated to bus connections between neighboring cells. The remaining I/O was comprised of control lines and status flags. A complete breakdown of the chip I/O is shown in Table 12.

TABLE 12
CHIP I/O DESCRIPTION

NO. OF PINS	FUNCTION
20	top cell interface bus
20	right cell interface bus
20	bottom cell interface bus
20	left cell interface bus
4	PE op-code control word
2	memory enable lines
1	PE error status flag
1	ALU overflow flag
1	100ns system clock

Chip Size and Speed

The final layout of the PE chip produced a core limited design. In other words, the size of the chip was limited by the logic contained within it and not by the number of I/O pads. The VTItools system was able to produce a chip size of 392 x 360 mils. This size is within the limits of 400 mils on a side that was given as the maximum practical size for a VLSI chip. The rather large amounts of memory defined the majority of the chip layout. Another factor in the large size of the chip was the high degree of bus routing due the eleven internal buses. If the size of the chip was determined to be too large for some reason, a reduction could be realized by limiting the number of internal buses. The removal of internal buses would cause a decrease in the ability of the PE to perform parallel operations and the PE's degree of

reconfigurability. However, as stated early the final layout is within the limits allowed by current VLSI technology. A diagram of the chip layout with dimensions given in mils is displayed in Figure 17.

The VTItools system was also used to determine if the final design would be able to run on the proposed 100ns system clock. A check of chip speed is necessary since it is impossible to arrive at an accurate number for wire delays until the chip is routed. A high level simulation of the chip model with capacitance added in for the calculated wire delays verified that the PE is capable of operating at the 100ns rate of the system clock. It should be noted that before this chip is ready for production a great deal of testing would still be needed to verify functionality of all the desired operations.

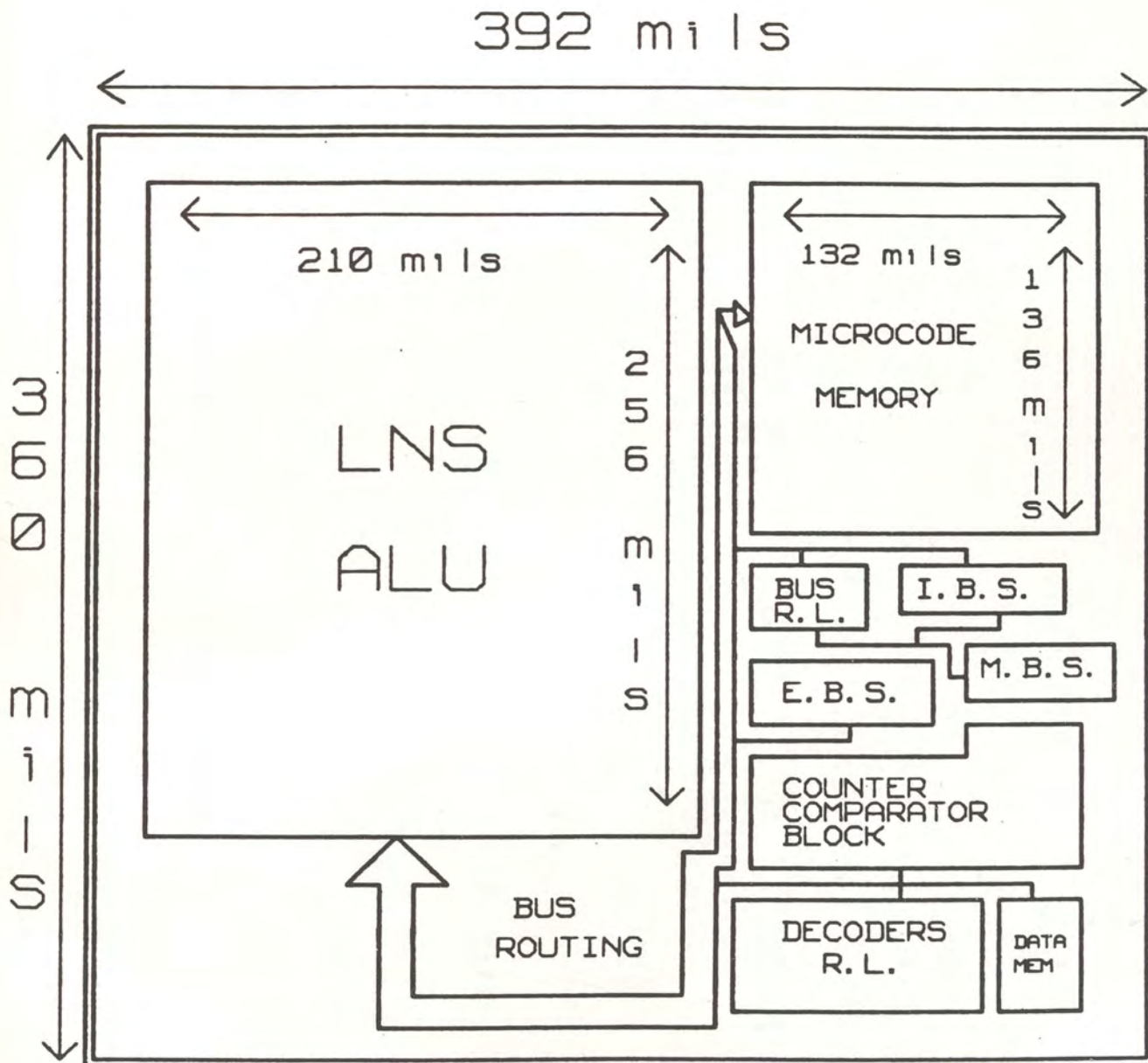


Figure 17. Chip Layout.

CHAPTER IV, CONCLUSION

The theory of optimal filtering, and especially Kalman filtering, has been extensively researched since its introduction in the early 1960s. The goal of this paper was to show how the recent developments in the area of logarithmic number systems (LNS) and very large scale integrated (VLSI) circuits could be used in the systolic array implementation of the Kalman filter.

A suitable systolic architecture was developed along with the implementation of the Kalman filter equations using LNS arithmetic. A processing element (PE) to be used in the systolic array was designed and its ability to perform Kalman filter related functions demonstrated. The only subjects yet to be covered are alternative PE designs and possible topics for future research.

Alternative PE Designs

There are always methods of improving an existing design; the question is what are advantages and disadvantages of the design change. The PE design could benefit from enhancements to its storage memory. The current design only allows one read and write operation per system clock. However, since the read operation occurs asynchronously it should be possible to perform two read operations from the same port in one system clock. This enhancement would be useful in reducing the number of clock cycles needed to perform various Kalman filter functions.

The extra memory operation could be implemented in a variety of ways. The negative and positive clocks could be used to generate different read enables. Another possible approach would be to subcycle the memory to allow the extra operations to be performed. A third method would use a 3-port memory to allow additional read and write operations, but would also require the addition of another memory bus and memory switch. All of the design implementations mentioned have their advantages and disadvantages. The design approaches presented are also not the only methods of allowing extra memory operations to occur. The use of the PE would have to be investigated in more detail to determine the most viable design enhancement.

Although a possible design change has been mentioned for the PE storage memory, that does not mean that it is the only design enhancement that may be useful. The designer of the systolic array in which the PE will be used should study the design and determine what other changes would benefit the array implementation. It should be noted that typically a design of this nature would go through many reviews by more than one designer in an attempt to produce the most useful end product.

Areas of Future Research

This paper attempts to give a good overview of the design for a PE to be used in the systolic array implementation of the Kalman filter. However, it is impossible to cover all the areas involved with the degree of detail that they warrant. Throughout the paper various areas that would benefit from future research were mentioned; however, there are three areas that should be stressed.

The first area of possible research concerns a new floating-point- to-logarithmic algorithm that would allow the accuracy of the result to be increased to 32 bits without increasing memory look-up table size. The new method would require the addition of 32-bit adders and some extra control logic. The use of this new algorithm and a redesign of the current PE could result in a 32-bit high speed ALU (Taylor 1983).

Another area of research concerns the implementation of the systolic array and the control structure that will utilize the designed PE. One of the more difficult Kalman filter operations that must be performed in the array is a matrix inversion. A new algorithm that uses the square root function has been proposed and should be investigated in detail. This algorithm should be very useful due to the relative ease in which the LNS ALU performs the square root operation.

The third area to consider for future research concerns the design of a compiler for the generation of PE microcode. A microcode compiler would allow users that are not extensively A compiler would also benefit the system designer by not requiring him to perform the actual coding process.

REFERENCES

- Abidin, Randolph L., "The Design of Standard Cell VLSI Circuits," University of Central Florida, 1984.
- 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.
- Hoffman, Tom. General Electric Company, Daytona Beach, Florida. Interview, 6 March 1987.
- 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.
- Kurokawa, T., Payne, J.A., and Lee, S.C., "Error Analysis of Recursive Digital Filters Implemented with Logarithmic Number Systems," IEEE Transactions ASSP, Vol. ASSP-28, December 1980, pp. 706-715.
- 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.
- Manno, Steven V., "The Application of Systolic Architecture in VLSI Design," University of Central Florida, 1986.
- 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.

- 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., Gill, R., Joseph, J., and Radke, J., "A 20-bit VLSI Arithmetic Processor based on the Logarithmic Number System," Department of Electrical Engineering, University of Florida and Honeywell Inc. Systems & Research Center, Preliminary Paper, 1987.