OPTIMAL DISTRIBUTED MICROPROCESSOR ARCHITECTURE USING MULTI-PHASE PROCESSING TO PERFORM A VECTOR, MATRIX MULTIPLICATION

By

LARRY GENE STOTTS

Bachelor of Science
Oklahoma State University
Stillwater, Oklahoma
1972

Master of Science
Oklahoma State University
Stillwater, Oklahoma
1977

Submitted to the Faculty of the Graduate College
of the Oklahoma State University
in partial fulfillment of the requirements
for the Degree of
DOCTOR OF PHILOSOPHY
July, 1979

OPTIMAL DISTRIBUTED MICROPROCESSOR ARCHITECTURE
USING MULTI-PHASE PROCESSING TO PERFORM
A VECTOR, MATRIX MULTIPLICATION

Thesis Approved:

Edward L. Ahrens
Thesis Adviser

J. K. Phillips

Jack A. Allen

Craig S. Sims

Norman N. Durham
Dean of the Graduate College

ACKNOWLEDGMENTS

I would like to extend my appreciation to Dr. Edward Shreve, chairman of my doctoral committe and my thesis adviser, for assistance throughout the course of my graduate education. To the other members of my doctoral committe, Dr. Jack Allison, Dr. Craig Sims, and Dr. Richard Phillips, I wish to thank them for their critical review of the manuscript and for their contributions to my education.

Over the past two years, several persons in addition to the committe members have given helpful comments and criticisms on the research described in this thesis. Hopefully, no one has been omitted from the following list: Dr. Gary Poffenbarger, Mr. Steve Hudson, Dr. Eugene Bailey.

Finally, I would like to thank my parents for their support and encouragement through my education in school, sports, and life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Figure Page

CHAPTER I

PROBLEM DEFINITION

Introduction

Significant alterations in engineering formulations and applications have emanated from the evolution of the digital computer. This has instituted the development of expensive machines of a general purpose structure capable of diverse operations. By nature, these instruments are large, slow, exorbitant, and complex. In later evolution, computer design technology produced faster, more efficient systems, but the units remained large and costly. The need for smaller, faster computers became apparent early in the computer age. This need instigated the use of special purpose computers, small in physical size, low in overall cost, and fast in execution time. The drawback to such special machines was the initial cost and limited application of the system. An example of a limited system of this design is the digital differential analyzer.

Neither large-scale systems nor small special systems appear to be suitable for real time applications such as vehicle navigation, signal processing, and digital filtering. To meet the requirements necessary to compute these algorithms efficiently, an alternative philosophy has emerged that utilizes the particular mathematical structure of a given class of problems to generate the computing system design. By designing the computer to take advantage of the structure of the problem, classes

of problems possessing similar characteristics may be processed efficiently in a special-purpose machine.

A group of problems whose mathematical structure can be used to generate the design criteria of the machine is the class of problems solved with vector, matrix operations. This problem structure suggests an array orientated machine capable of parallel operation. In the majority of designs this organization has resulted in an array processor composed of identical processing elements with an effective interconnecting structure. Some examples of this class of problems are recursive linear filters, vehicle nagivation, phased-array radar control computations, and sonar receiving array data processing.

The design and implementation of such a computing apparatus will be affected by the recent advances in integrated circuit technology. The development of large scale integrated circuit technology has prompted the fabrication of complex digital processors on a single substrate. The current advances in integrated circuits, as well as possible future advances, must be taken into account in the design of a computer system. In the next chapter a survey of array structured systems research, as it pertains to the design of special organized computers, is presented.

Problem and Approach

The designs of special-purpose computer systems for the evaluation of vector, matrix products have thus far been constructed using arrays of identical processor elements functioning at a common cycle time. These machines are composed of N^2 processing units, interconnected by some type of bus structure and manipulated by a controller. In the event that the array to be processed is of dimension N^2 or less, maximum throughput is

maintained. However, in situations resulting in an array of dimension greater than N^2 or less than N^2, the efficiency of machine operation is reduced due to processor idle time. Processing elements may be added to the array to meet larger array requirements or removed to match smaller arrays, resulting in the dimensions of the data array and processor array being made equal.

The primary restriction of this architecture is illustrated by placing a time, power, and size constraint on the system. As the dimension of the data array is increased, the design quickly overruns the constraints. In this situation, if the number of processing elements is made equal to the data array by an integer multiple, efficiency can again be attained. This is true only if the processor operation time is fast enough to perform all the computations in the required time frame. In most cases it is impossible to provide a processor array related to the data array by an integer multiple.

It is the concern of this research to obtain an optimal architecture capable of computing vector, matrix operations. The architecture will be optimized to cost and constrained to time, power, and circuit size. A flow chart of the problem is shown in Figure 1. The approach will employ the use of two or more processors that operate at different cycle times. It is evident that processing speed is directly related to power and cost, with an inverse relationship to circuit size. A design founded on these relationships lends itself to linear integer programming optimization techniques. Optimization based on cost of the array elements will result in an efficiently designed computer capable of adhering to time, power, and circuit requirements. The approach flow chart is shown in Figure 2. The results of this study will produce an algorithm to follow in the

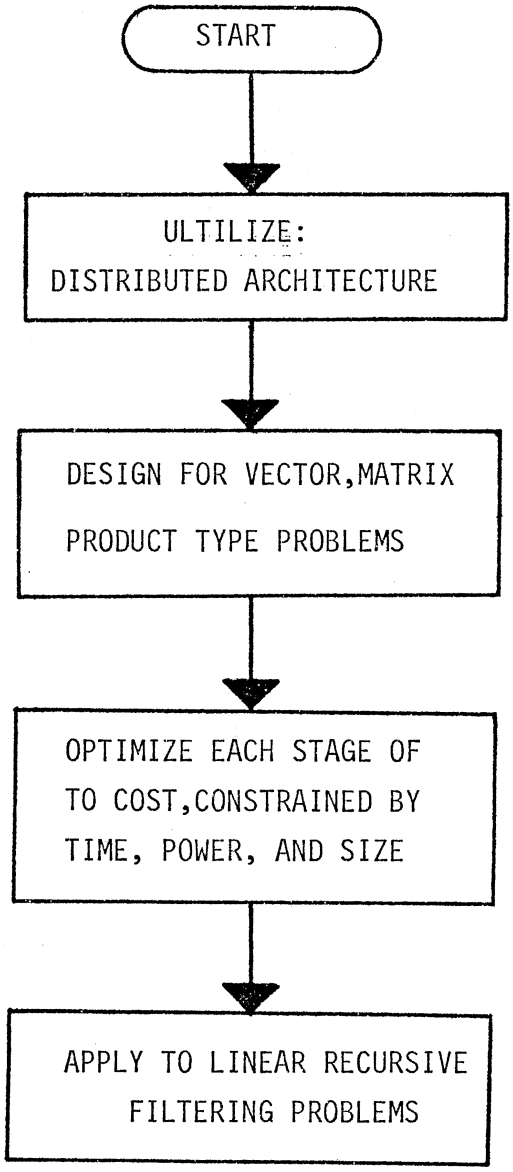


Figure 1. Problem Flow Chart

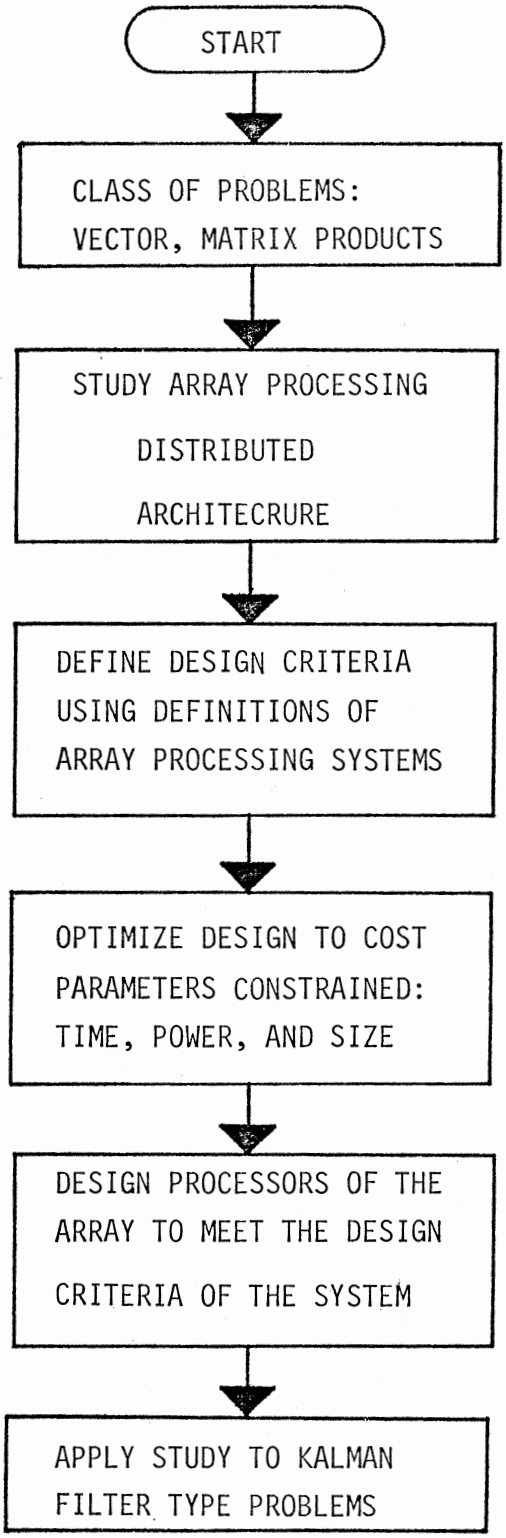


Figure 2. Approach Flow Chart

design of computers to perform vector, matrix computations. This procedure will yield an optimal structure capable of accomplishing the desired computations in a specified time period, optimized to cost and constrained to power, time, and circuit area. A second product of the work will generate the interconnecting circuits necessary to interleave the computed data to the proper accumulators during the calculation process. In the course of this research, two other conclusions will be reached. In the area of floating point hardware, a maximum throughput structure will be illustrated. Further, a design of the most efficient bus structure of transferring the required data will be analyzed. This work applies system theory techniques to the heretofore "black magic" solution to the design of digital systems.

CHAPTER II

SURVEY OF DISTRIBUTED PROCESSOR SYSTEMS

Introduction

A study of available literature on special computer systems produces a number of distributed processor-based designs. Several pertinent computer structures are discussed in the following sections.

The Unger Computer

One of the earliest examples of a distributed architecture computer was proposed by Unger (1). This system used a stored program to handle specific problems by directly processing information in planar form without format conversion or scanning operations. The structure of the device lends itself to handling pattern detection problems.

The structure of the Unger system is illustrated in Figure 3. This computer uses a master control unit and a rectangular array of processor elements. Each processor element can communicate with the four adjacent elements and receive commands from the master control. The controller is composed of a random access memory to store instructions, decoding circuits, and a clock. Commands from the controller are generated in parallel to all processing elements, but individual processing elements are not addressable. Programming is accomplished with 14 assembly language instructions that are executed by the controller.

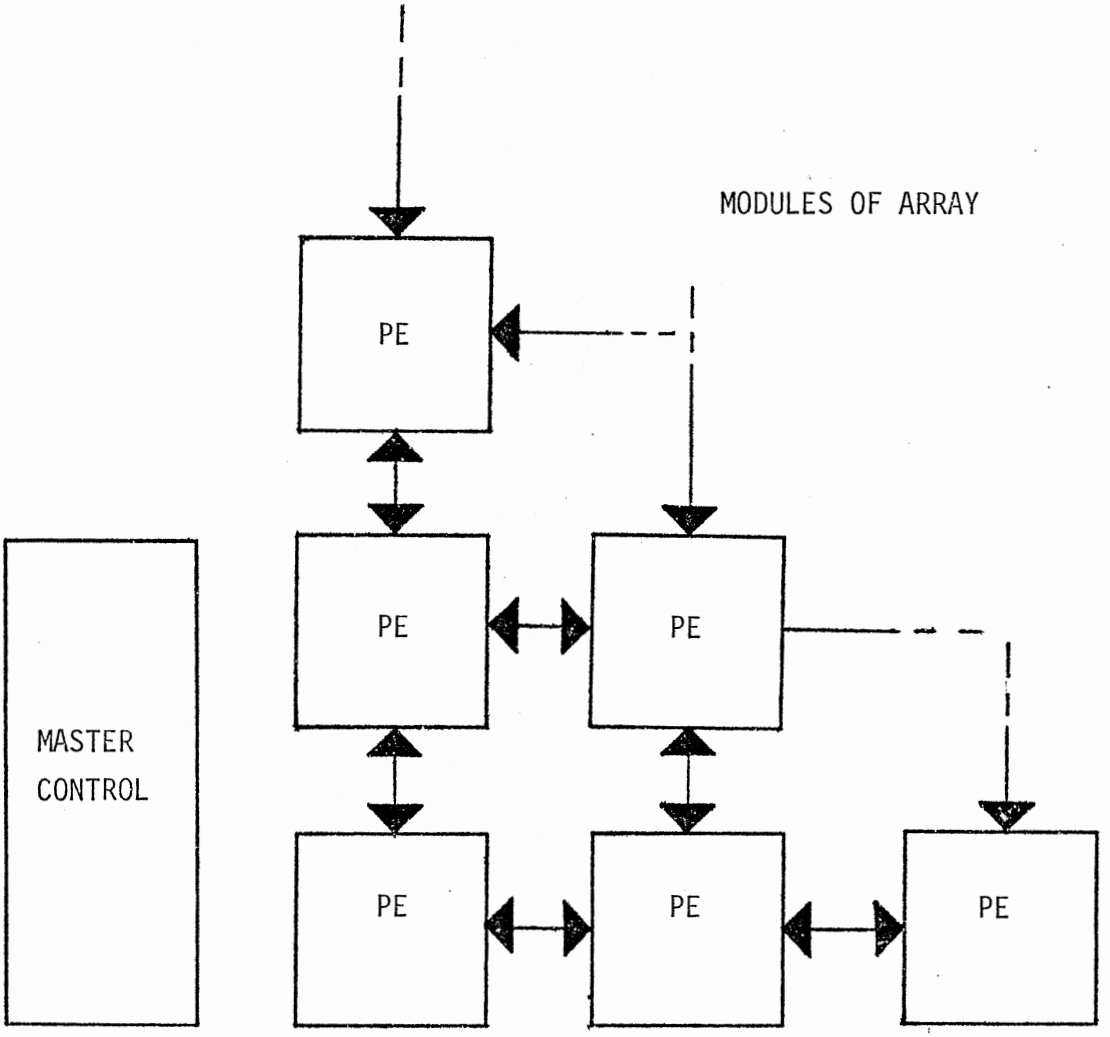


Figure 3. The Unger Computer System

Each processing element consists of a single bit accumulator, some associated logic, and a small random access memory. Inputs to each processing element consists of control lines from the system controller and links to the accumulators of the adjacent elements.

A branch on accumulators equal zero is accomplished in the control unit by using a logical adder to evaluate the inputs to the controller from the accumulator of each processing element. This instruction functions in the same way as the conditional transfer used in conventional computers by causing the control unit to skip the next instruction when zeros are detected in the accumulators. This transfer instruction composes the only decision dependent command utilized by the machine.

The Holland Machine

A computer system organization has been described by Holland (2) that places control at the processor element level in the system. This system organization is in direct contrast to the central control concept proposed by Unger.

The concept of the Holland machine provides a basis for investigation of the theory of automata and computability. Holland's system consists of a two-dimensional array of identical processing elements, with each element containing a storage register, routing logic, and auxiliary registers. During any given machine cycle, a processor element is either active or inactive. If the element is active, it decodes the contents of its storage register as an instruction and proceeds to execute the operation. Following the execution of the instruction, the processor element passes its active status to the next element, which may be any adjacent processor element in the array. Using this concept, sequences of

instructions are scattered throughout the array of processor elements, with an arbitrary number of instructions being executed at any given time.

There are three phases that compose the operating cycle of this system. During the first, processor element storage registers may be set to values introduced by external sources. In the next phase, all active elements determine the address of their operands by logically enabling data paths. The last phase consists of the execution of the instruction in the storage register.

The disadvantage of this machine appears to be the difficulty of programming in an efficient manner that will allow a large number of processor elements to be active during a machine cycle. Also, it is necessary to use a massive amount of hardware to solve a reasonable computation problem.

The important contribution of this work is the development of an array of locally controlled identical processing elements. The hinderances of this approach are the extensive amount of hardware utilized and the programming difficulty.

The Comfort Machine

The array-structured system based on the concept of local control was further studied by Comfort (3). This study culminated in a modified Holland machine with a fixed-size rectangular array of processing elements. Processor elements are composed of two relatively independent sections: the control section and its memory, and the communication section. The arithmetic units are placed beside the array of processor elements in this configuration and perform all mathematical and logic operations. This computer contains no central control unit; therefore, each processor

element executes its own program once it is enabled. The execution of a set of instructions causes the enabling and disabling of successive processor elements.

Comfort's computer was designed to provide some improvements to the Holland computer, which are listed below:

1. System is easier to program by several orders of magnitude.
2. Machine size is reduced by a factor of five.
3. Utilization of hardware is improved by a factor of three.

The drawback to Comfort's system is that only one program sequence per arithmetic unit can be operated concurrently.

The SOLOMON Machine

The SOLOMON (Simultaneous Operation Linked Ordinal Modular Network) system is a distributed architecture computer introduced by Slotnick, Borck, and McReynolds in 1962 and later revised in 1966 (4). The architecture was conceived to satisfy a particular class of problems and adhere to current needs in computing capabilities. The primary purpose of this device was to implement matrix operations and computations. This class of problems consists of linear systems analysis, matrix calculations, and solutions to systems of ordinary and partial differential equations.

Figure 4 illustrates the construction of the machine and its three major units. The network control unit (NCU) is first and provides the central control of the machine. The NCU is composed of at least one arithmetic and control unit, and is expandable to a multiple unit configuration. An array of processing elements (PE) in a 32 x 32 structure makes up the second major unit. The processor configuration is designed to allow modules of 256 PE's with the associated memories for each to be added or

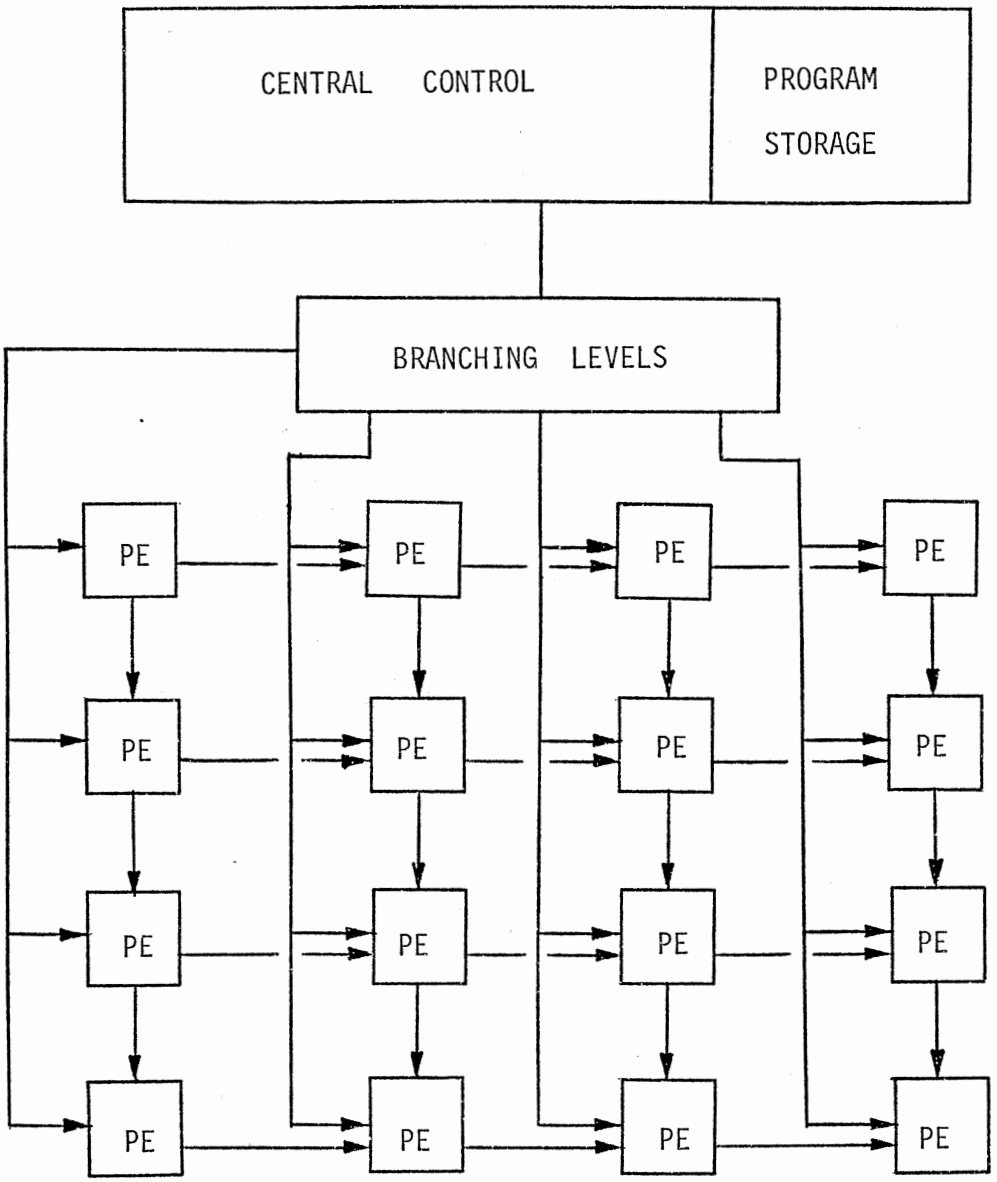


Figure 4. The SOLOMON Computer System

removed from the unit without design alterations. The third major unit is the input-output unit (IOU) which is comprised of five modules of 32 data channels, with each channel acting as a separate input-output link.

The processing elements of the computer are identical and each possesses complete arithmetic capacity. An individual processing element has associated with it two memories with 4096 bits of storage (expandable to 16,384 bits) in each memory. A processing element can perform serial logic and arithmetic operations and can communicate serial data to the four adjacent elements in the array. The elemental conclusions of this study reveal the adaptability of a machine that utilizes identical processing cells under a central control and the capability of serial communication with the four nearest adjacent elements.

The Gonzalez Iterative Computer

A multilayer iterative circuit computer (ICC) has been proposed by Gonzales (5) and is an improvement on the work done by Unger and Holland. The architecture of this computer provides the capability to solve problems involving spatial relationships between variables. The processing elements in this architecture are placed in three stacked layers. The layers are identical and consist of a program layer, a control layer, and a computing layer. The data and instructions are stored in the program layer; the control layer performs the decoding of instructions that are executed by the computing layer. The programming sequence is similar to the Holland computer in that each instruction specifies the processing element that contains the next instruction. Pipelining allows the control and programming layers to work on the next two instructions during the execution phase.

The elements in the three planes of M x N modules are identical and are allowed to communicate by means of control lines. The internal architecture of the elements is composed of an accumulator, a register, a decoder, and a number of switching matrices used to interface the decoder to the data and control lines. As with other computers of this type, the programming is difficult and the hardware is inefficient. The major features of the Gonzales design that are of further interest are

1. The path connecting method that retains the time access features of a common bus computer, while allowing simultaneous operation of other paths in the system.

2. The complete separation of control signals from the data flow.

3. Three-phase operation, with each phase active simultaneously on each layer, but executing different instructions.

The ILLIAC IV Computer

A distributed architecture system called the ILLIAC IV was introduced by Barnes, Brown, Kata, Kink, Stokes, and Slotnick (6). The computer is a continuation of the work done on the SOLOMON computer and is employed to implement matrix, vector computations.

The design of the ILLIAC IV system is illustrated in Figure 5. The processing elements (PE) are placed in four arrays of 64 elements each with one control unit for each array. The function of the control unit is to decode instructions and control the 64 elements in array which it is designed to manipulate. The operations of the four arrays can be combined to perform multiprocessing or single processing operations, all under control of one program. The system program is stored in a general-purpose computer, a Burroughs B6500, that is responsible for loading the

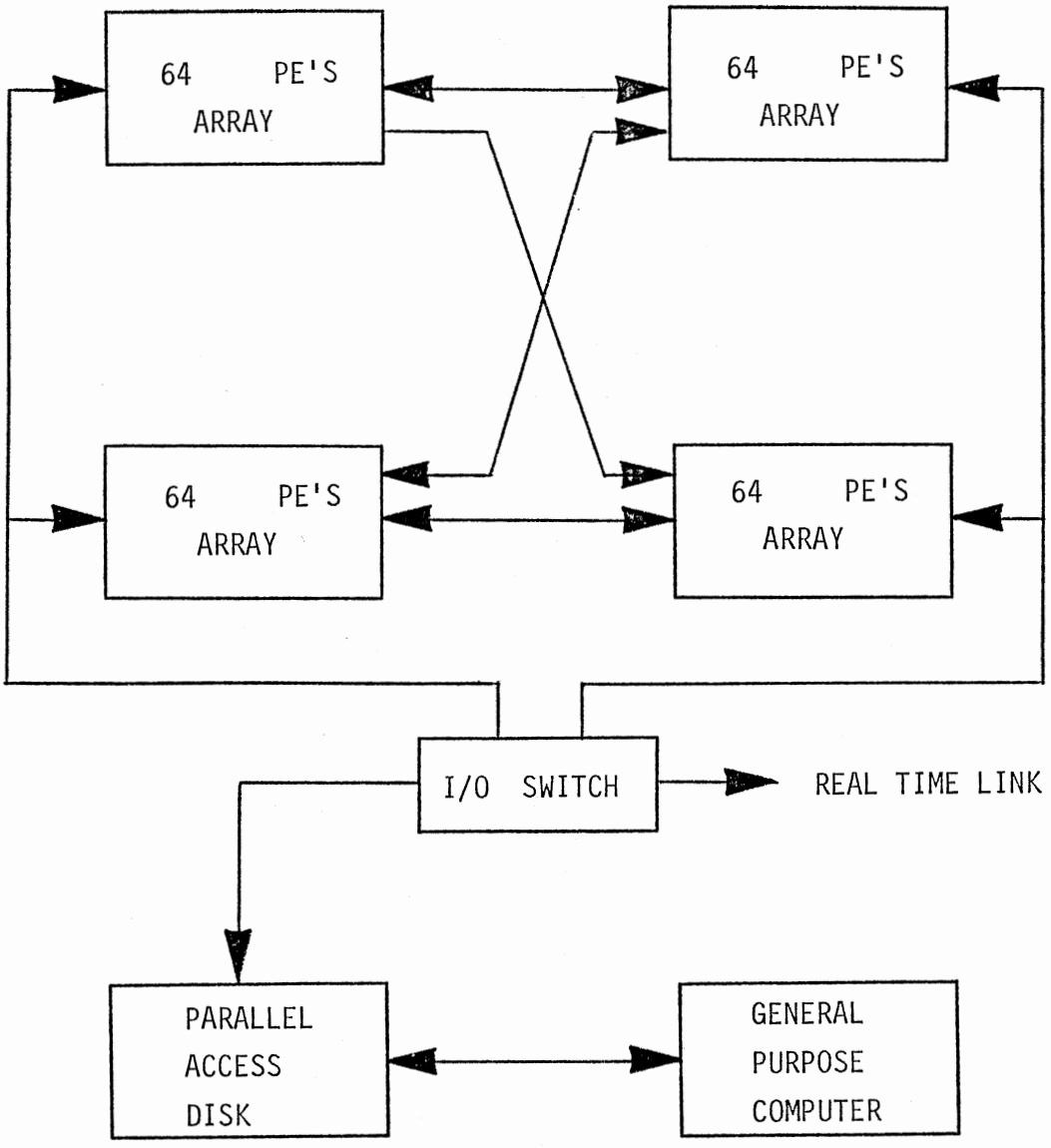


Figure 5. The ILLIAC Computer System

arrays and controlling the configuration of the system, and outputting the data. Backup memory for the array is provided by disk in a parallel access configuration that is directly attached to the ILLIAC IV system.

The architecture of the ILLIAC IV consists of four SOLOMON arrays of 64 processors each to provide 256 processor elements. A processor element in the system can perform 240 nanosecond addition and 400 nanosecond multiply operations on a 64-bit operand. The processors are each constructed of 10^4 ECL gates and a memory with 240 nanosecond delay and a 2K word configuration.

The like units of the ILLIAC IV are constructed to be interchangeable, as are the power supply parts. Trouble-shooting is thus made easier and down-time is reduced.

The MCB Machine

The Modular Computer Breadboard (MCB) was introduced by the NASA Electronics Research Center (7). The architecture is in the form of a modular system that can be reconfigured to operate as a distributed network of processors. The system can be laid out in the form of columns of processor elements that may be unplugged if not required in the computation. Each individual processor may be operated independently of the others or in conjunction with others, if required.

Four modules compose the structure of the system: a memory, control unit, arithmetic unit, and input-output unit. Alterations to the configuration are accomplished by the configuration control unit and the configuration control switches. Prime importance to the configuration capabilities is the use of triple-redundance to provide extra reliable

operation. The modules of this computer are constructed of LSI circuits and emphasis is placed on plug-in type units.

Berkeley Array Processor

The Berkeley Array Processor was introduced in 1970 by Dere and Sakrison (8) to be used as a general-purpose system. The processor will efficiently perform the operations of convolution, correlation, recursive filtering, matrix multiplication, and fast Fourier transform.

Dere and Sakrison's system functions as an input-output device, operating in conjuction with an IBM 1800 computer. In this design, the program and data are stored in the IBM 1800 and accessed under control of the array processor. Arrays of data stored in the IBM 1800 are transferred to the 104-word shift register memory of the array processor via a pseudo two channel data link. The structure of the array processor consists of shift register memory, an array index unit, arithmetic section, accumulator, and necessary functional control logic. The clock period is 140 nanoseconds, and the data paths and registers are constructed to accommodate 16-bit words.

Instructions for the processor are composed of 17 operations capable of handling complex data processing. The majority of the instructions are utilized to control the arrays of data and to perform bookkeeping operations on input and output information. The input-output operation of this processor introduces the concept of storing data in a larger computer and processing the data in a special central processing unit designed specially for the purpose of array structured data. This system uses only standard design techniques in its construction and provides some significant logic innovations.

The Cannon Computer

Efficient performance of matrix operation was the main design constraint of the array processor proposed by Cannon (9). The system was intended for use in implementing algorithms utilized in linear recursive filters and to extend the designs of the SOLOMON and ILLIAC IV computers.

The structure of the system, shown in Figure 6, consists of a two-dimensional square array of processor units and a global control unit. Control of processing elements is maintained through the use of parallel control lines to each of the identical processor elements. Besides the control logic, the global controller has arithmetic hardware and a large data storage capability. This design allows the control unit to not only decode and control execution of instructions, but further allows for data processing to take place in the controller.

The processing elements are identical in structure and consist of sixteen 32-bit words of memory, a floating point adder-subtractor, floating point multiplier, and logic to control the data flow. The design calls for all processing elements to receive identical control signals and perform the same operations during each cycle of computation. The matrix data is stored in the memory of the processor element array in the same (i, j) element location that the data holds in the data array. Processor elements in the array are interconnected by singular horizontal, vertical, and diagonal data lines that allow special data transfer operations to be implemented. These special operations are broadcasting, rotating, skewing, and transposing the data matrix.

The structure proposed in this design yields a significant reduction in processing time over conventional machines due to its parallel processing

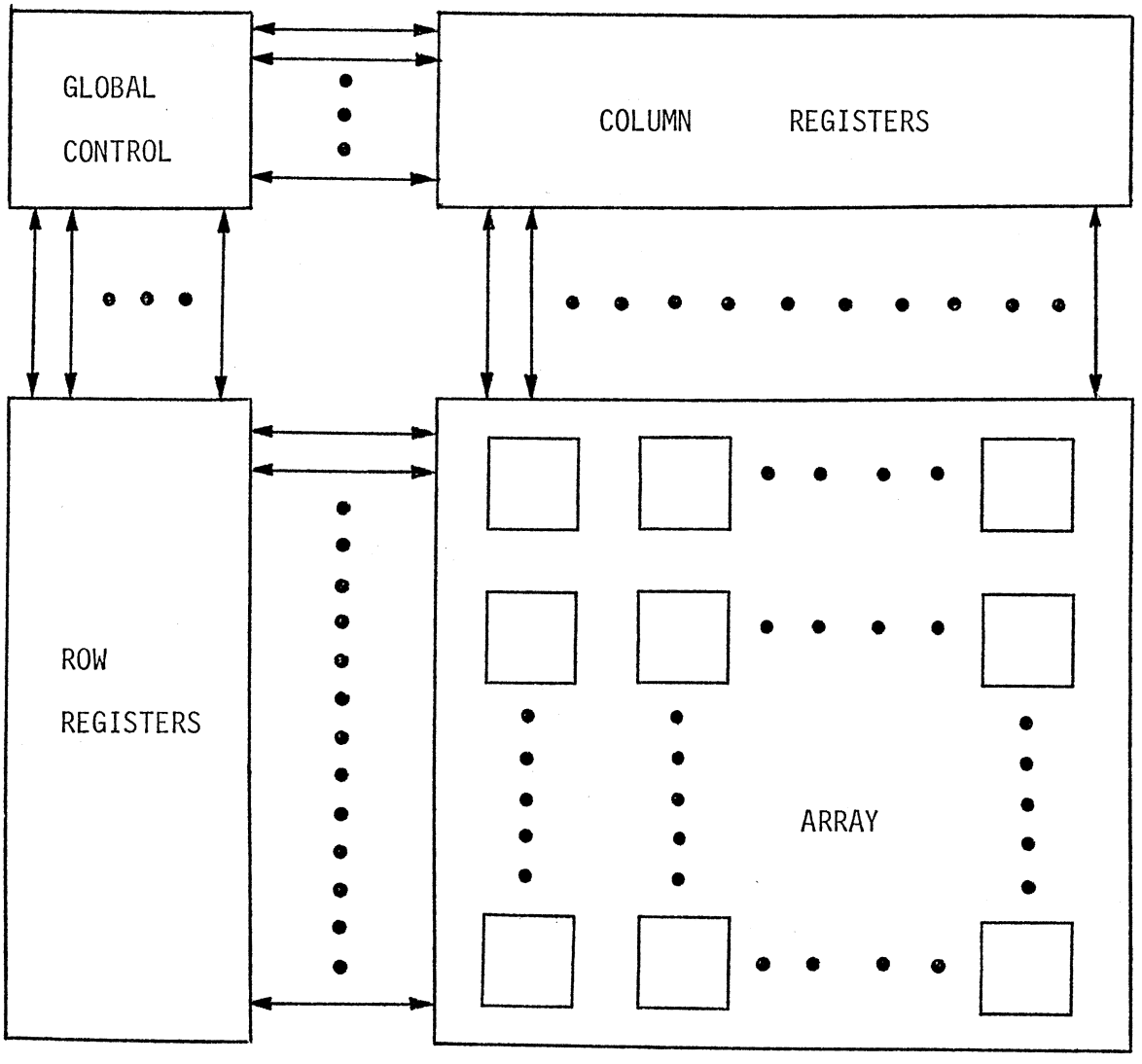


Figure 6. The Cannon Computer System

capability. A disadvantage to this scheme is the need of N^2 computing elements to compute data in an N x N dimension array. The system can handle efficiently an array that is of dimension N x N, but for larger or smaller arrays, it appears to be less effective.

The General Electric Matrix Processor

This system is an implementation of the Cannon computer designed and built by Moyer, Rice, and Fifolt in 1977. The controller used was a Z80 microprocessor and the processor element consists of hardware floating poing units constructed in an 8 x 8 array. This architecture is shown in Figure 7.

Real time linear recursive problems are the prime purpose of the machine, and in particular the Kalman filter algorithm is easily handled by the system. In its full operational state this unit requires one processor element for each element in the data array. Increasing the capability of the system requires the addition of more processor elements.

The techniques utilized in this system are of interest since the implementation was realized using LSI circuits. There are 400 integrated circuits making up the system, and it operates at a 60 watt power level. The system is capable of computing all the matrix functions commonly found in real time control and signal processing applications. Complex matrix algorithms may be computed using the microprogram capability of the system which allows it to function in time limits that cannot be met by conventional computers.

Summary

The distributed architecture design techniques proposed over the

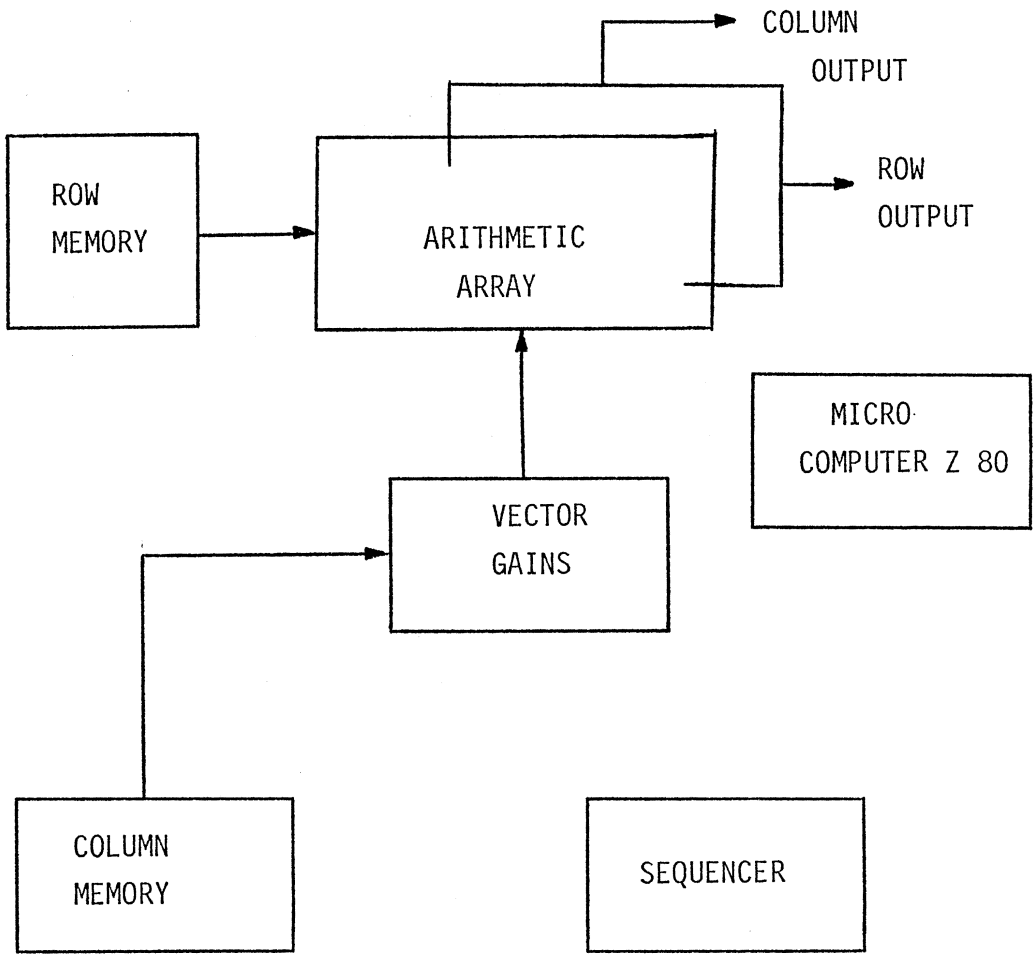


Figure 7. The General Electric Processor

last 20 years have been presented, along with advantages and disadvantages resulting in these systems. A growing need for small special-purpose computers has also been discussed and various problems such as filtering and navigation have been pointed out to have array structures. It has been deemed advantageous to base the design of the hardware upon characteristics of the problem to be solved in situations where a group of problems have mathematical structural similarities. Research in this area is being caried out in both industry and university environments with both entities placing strict concern on advances in LSI technology. The ultimate culmination of these studies will be algorithms capable of designing fast, low cost, and efficient systems able to handle the myriad of problems that face the engineering community.

CHAPTER III

DEFINITIONS OF DISTRIBUTED ARCHITECTURE AND PROCESSING

Introduction

Distributed or parallel processing is defined as processing two or more portions of an algorithm by two or more processing units during the same interval of time (10). This processing takes place at the task, subtask, instruction stream, or data set level. The result of this definition generates a multiple processor system organization in the hardware (11). Once the hardware is defined as a multiple processor system, it is further divided into two processing structures. These structures are termed single instruction multiple data (SIMD), shown in Figure 8, and multiple instruction data (MIMD) (12), shown in Figure 9.

A definition based on the organizaiton of the hardware processors alone does not define the entire concept of distributed architecture (13). The definition must consider the concepts under which the distributed processors are allowed to communicate and share memory space. Further, the programs to be executed by such a distributed system should make maximum use of the parallel capabilities of the computer.

SIMD Architecture

This architecture is sometimes referred to as a parallel processing system (14) that uses one control unit to fetch and decode the program

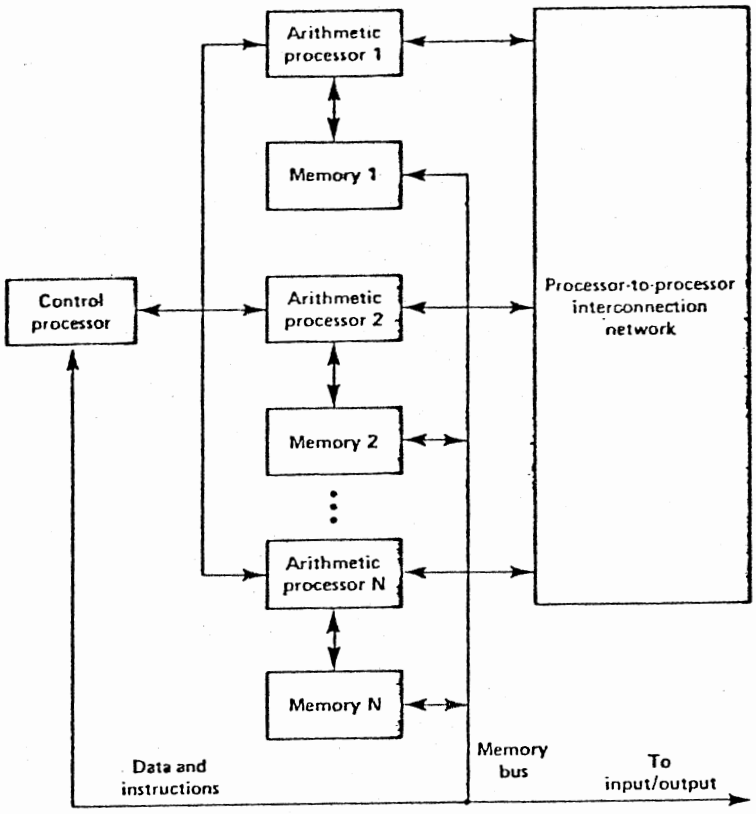


Figure 8. SIMD Architecture

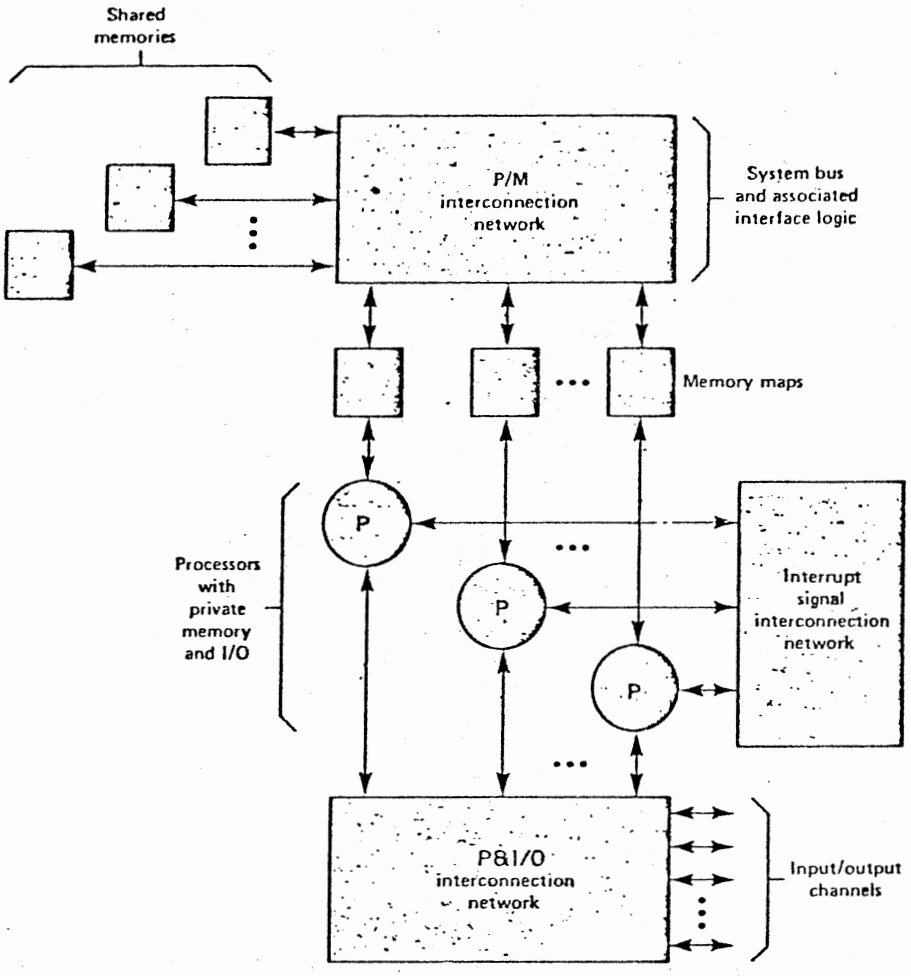


Figure 9. MIMD Architecture

instructions. The instructions may be executed in the central control unit or they may be sent directly to subordinate processors in the system.

The SIMD systems are further divided into three subclasses of this architecture and called array processors, processing ensembles, and associated processors (10). The array processor is designed to allow the instructions to operate on vectors of data at the same time with the control unit having only limited capabilities. To implement the processing ensemble, the control unit must be a complete computer and the processing units are only allowed to communicate by passing data through the control unit. The last subclass is that of associative processors, which are designed to allow subelement processors to access and operate on data only by its content and not by storage locations (10, 11, 12).

Array processors are considered to be the most applicable subclass of SIMD architecture from cost and throughput criteria (10). The need for maximum throughput motivates the design of array processors since their tremendous throughput is accomplished by simultaneous operation of individual processors on different streams of data. Three criteria must be met to allow an array processor its maximum parallel capabilities (10). First, all calculations should be described by vector instructions which cause large amounts of data to be manipulated simultaneously by one operation. Secondly, there must be high speed data paths between processor elements; and last of all the block of data that is to be processed during one time interval must also be fetched from memory in one time interval. Failure to meet these criteria will result in the system tending toward serial operation and a major reduction in throughput (10, 12, 13, 14).

MIMD Architecture

The vector operations of SIMD are not used in an MIMD system, where parallelism is obtained by performing different operations on individual data sets in a given time interval (10). The results of the separate operations are combined to form an end result to the computations. The primary criteria for efficiency of an MIMD system is the design of proper synchronization of the individual subsystems and allocation of the processing in an effort to balance the computational load on the system. The SIMD system operation does not face the same synchronization problem since each individual processor of like type is doing the same operation concurrently. The MIMD architecture may be divided into two subclasses, which are called the multiprocessor system and distributed system (10, 12, 13). The multiprocessor system uses one controller as a master to allocate the slave processors individual tasks as requests for these tasks are required. The slave processors may or may not be capable of all doing the same functions to the incoming data stream (12, 13, 14).

This multiprocessor system may be used in applications of general purpose computations, with each slave processor performing general tasks. As new task requirements are generated, an idle processor is used to perform the operation or to pick up another processor's work if a failure occurs in some other subordinate unit (10). The system may be altered to allow different processors to do separate tasks, but as new programs of the same type are generated, they must wait for a specific processor to become available. In certain environments only a few of the slave processors would be used while the remainder go idle (10, 13, 14).

MIMD architecture has a second subclass called a distributed system (12) which is composed of multiple processors designed to do specific functions in a partitioned system. With this architecture the subordinate processor systems may be located in one area, or separated by large distances and connected by communications networks. The algorithms to be executed on such a system must be known prior to operation of the system so that software may be segmented into dedicated programs to drive the individual processors. Such a diverse system, however, reduces interaction between sybsystems and makes debugging of individual systems less complicated (10, 12).

Communication in distributed systems consists of messages sent between processors or blocks of data being transferred from one unit to another by way of shared peripherals or serial communication channels. A further specification of this subclass of architecture may be drawn from the requirement that the memory of the system not be shared by any of the sub-units (12). From these definitions it is evident that the disadvantages of such an architecture are (a) load characteristics are difficult to determine for general program usage, (b) poor parallel usage of subprocessors, and (c) general difficulties in controlling the system if expansion of the system is desired (10, 13, 14, 15).

Coupling

The MIMD subclass of multiprocessor systems has been further classified by the amount of memory shared between its subprocessors. Systems are referred to as being tightly coupled or loosely coupled. A tightly coupled system is designated as one that is subjected to a strict control scheme implemented in self-contained hardware (16). Tightly coupled

systems have been defined by Bowra (11) to include processors which transfer data through shared memory. However, this definition is not followed by Weissberger (10), and Enslow (13) in their concepts of multiprocessor systems.

Loosely coupled systems are considered to be systems that have no interaction between processor programs but do allow memory to be shared (10, 13). Systems of loosely coupled processors require adequate communication and memory sharing to reduce the ramifications of subsystem failures. These results are obtained by alloweing dynamic reconfiguration of the operating system in the event of subsystem failure. Communication capabilities will also reduce the interaction of processors during simultaneous attempts to acquire data in shared locations. Such control is accomplished by allowing a priority criteria to be established (10, 11, 13, 14).

Definition of Time and Space Complexity

Once an algorithm is in a form ready to be executed by a computer, two questions must be answered concerning its complexity (17):

1. What is the extent of memory space necessary to execute the algorithm?

2. What is the time required for execution of the algorithm?

To attempt to answer these two questions it is desirable to utilize some algorithm evaluation criteria. The object is to determine the dependence of the time or space required to solve the problem as it grows in order and observation. It is necessary to associate with the algorithm an integer, called the size of the problem, assumed to be a measure of the input data and order of the system.

The time required for solution of an algorithm may be expressed as a function of size of the problem, and called the time complexity (17). The limiting behavior of the complexity as size increases is called asymptotic time complexity (18). An analogous definition can be made for space complexity and asymptotic space complexity (18).

The asymptotic complexity of an algorithm is said to determine the size of the system which can be solved by the algorithm. An algorithm may process incoming data of size N in time TN^2 for some constant T. From this it is seen that time complexity of the algorithm is order N^2. A better definition (18) may be generated by stating that a function g(N) is of order f(N) if there exists a constant C such that $g(N) \geq Cf(N)$ for all but some finite set of non-negative values for N.

It is suspected that increases in throughput of data brought about with each new generation of digital computers would decrease the concern over efficient algorithms or more efficient architecture. However, as computers increase in throughput, and bigger problems can be solved, it is still the complexity of the algorithm that limits the increase in problem size that may be handled by a faster computer.

To further illustrate this time complexity definition, consider the case of five algorithms to solve the same problem, where each has a different time complexity (19):

Algorithm	Time Complexity
A_1	N
A_2	N log N
A_3	N^2
A_4	N^3
A_5	2^N

The definition of time complexity used here is the number of time units requried to process an input data set of size N. For example, one unit of time equates to one millisecond; therefore, A_1 may be processed in one second with an input size of N = 1000 and A_5 may be calculated in one second, but the input size is N = 9. Calculating the size of N relative to one second, one minute, and one hour will give the values shown in Table I (18).

TABLE I

LIMITS ON THE SIZE OF ALGORITHMS

Algorithm	Time Complexity	One Second	One Minute	One Hour
A_1	N	1000	6×10^4	3.6×10^6
A_2	N log N	140	4893	2.0×10^5
A_3	N^2	31	244	1897
A_4	N^3	10	39	153
A_5	2^N	9	15	21

As computer components become faster, and if a ten-fold increase in calculation speed is assumed, it is possible to calculate another set of results for Table I. Table II (18) gives the size of a problem which may be processed as a result of a ten-fold increase in data processing speed.

TABLE II

EFFECTS OF TEN-FOLD SPEED-UP

Algorithm	Time Complexity	Problem Order Before Speed-Up	Problem Order After Speed-Up
A_1	N	l_1	10 l_1
A_2	N log N	l_2	Approx. 10 l_2
A_3	N^2	l_3	3.16 l_3
A_4	N^3	l_4	2.15 l_4
A_5	2^N	l_5	l_5 + 3.3

The results of Table II illustrate, for example, that algorithm A_3 may handle data 3.16 times larger with the ten-fold increase in computer speed, but A_5 is increased by only 3.3 added to its previous size.

It is now thought that the rate of increase in computational throughput due to technology improvements is declining, which suggests that further increases in throughput will only result from better algorithms or systems architecture. Assuming that an algorithm is fixed, the most effective method of increasing the throughput is to use distributed architecture, or in other words, process the data in parallel whenever possible. Most conventional computers operate in a strict sequence with only one operation taking place at a given time, called serial processing. The distributed architecture approach replaces a computation requiring N steps by m independent subcomputations occuring simultaneously. Not all

algorithms adapt well to parallel processing, so overall results of a fixed parallel architecture as related to an open class of algorithms is not thought possible (18).

Summary

The intent of this chapter has been to introduce the distributed architecture configurations and tabulate some of their advantages and deficiencies. The background literature on the dilemma of vector, matrix structured computation has always been directed towards the use of single instruction, single data path architecture. It is pointed out in this chapter that the idiosyncrasies of array processing are far more applicable to this research than the other possible configurations.

Having once analyzed the architecture definitions, the concept of time complexity is viewed. This time complexity definition will serve as a primary constraint to the optimization problem and provide a basis of design for real time computations. The steps in computing a vector, matrix product will provide a fixed algorithm to be used in solving the problem. By fixing the algorithm, the architecture will be allowed to vary in order to meet the time complexity and other constraint criteria.

CHAPTER IV

OPTIMIZATION OF DISTRIBUTED ARCHITECTURE

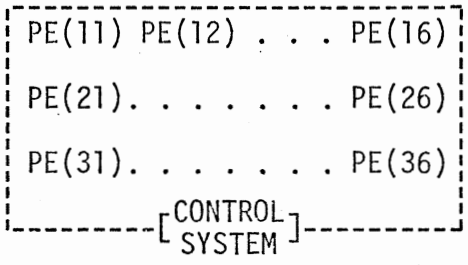
Introduction

In the interim of the initial research and conclusion of the literature survey, array processing formulas were employed to realize vector, matrix multiplications and similar mathematical operations. In each instance, the definition of array processing was closely followed and led to a computing structure composed of elements exactly equal to the dimension of the problem array.

An example of a matrix, vector multiplication problem is:

$$\begin{bmatrix} X_{11} & X_{12} & X_{13} & X_{14} & X_{15} \\ X_{21} & X_{22} & X_{23} & X_{24} & X_{25} \\ X_{31} & X_{32} & X_{33} & X_{34} & X_{35} \end{bmatrix} \begin{bmatrix} Y_1 \\ Y_2 \\ Y_3 \\ Y_4 \\ Y_5 \end{bmatrix}$$

An array processor element configuration is:



One processor element (PE) for each term in the matrix

The solution of an M x N matrix, vector product requires (M·N) multiplications and M(N-1) additions. Verification of this conclusion is provided by Aho, Hopcroft, and Ullman (18) in a general theorem of matrix multiplication algorithms. Their theorem illustrates that, in general, there exists no procedures to diminish the aggregate multiplications and additions fundamental to the solution of a vector, matrix product, such as $\overline{X}A$, where $\overline{X}$ is a matrix of order M x N and A is a vector of order N x 1. M times N multiplications and M(N-1) additions represent closure of the algorithm and approaches the assumption of a fixed procedure of lowest order. With this assumption made, attention is then concentrated on determination of an architecture, variable in structure, optimal in cost, and constrained by time, power, and size.

The literature study centers around an array mechanism composed of M x N operational units, interconnected to allow each unit to compute one partial product of the matrix, vector product. This design criteria stipulates that as the dimension of the problem matrix increases or decreases, the dimension of the computing mechanism must alter to equate to the order of the problem. The variations in array design are applicable to machines of an unlimited category; however, if real time design constraints are imposed on the structure, alterations must be generated to correct the design.

Array Processing Limitations by Definition

The definition of SIMD processing stipulates that individual processors must manipulate divergent data streams simultaneously. A conjectural criterion predicated by the Array Processor subcategory necessitates the concurrent execution of the same sequence of instructions under one

direct controller unit. Further definition infers that the data paths interconnecting the processing elements be of a high speed parallel nature, that all data fetched in one time period be processed in one time period and stored in one time period. A listing of the design parameters for an array processor is as follows:

1. Processors should process different data streams concurrently.

2. Processors should execute the same instruction concurrently under one control data stream.

3. All data paths should be high speed system.

4. Individual data blocks must be fetched in one interval T_1, processed in one interval T_2, and stored in one interval T_3.

5. The number of operations occurring simultaneously in individual modules must be maximized.

Design Limits of Array Processor

If the criteria and structure of the array processor is utilized in the presence of constraints such as time, power, cost, and circuit size, system performance is seriously degraded. The imperfections pertinent to the circuit design of array processors are best illustrated by attempting to design a processor array for a constrained problem. Time, power, and size restrictions must initially be designated and thereby provide limits to the choice of hardware capable of performing to the specifications.

Array Processor Design Example 1

Design Specifications: Data Time--1 microsecond
System Power--10 watts
(Multipliers)
Circuit Size--20 square units

Processor Chosen for Job (Specifications):
Cycle Time--600 nsec
Power Per Unit--1 watt
Area (Multiplier)--1.5 square units

Array Design Data:
10 Multipliers--10 watts power
Cycle time is less than 1 μsec
System Area = (10 mply)·(1.5 sq units/mply)
= 15 sq units

*Maximum processor array size is 10 processing elements.

*Maximum number of matrix terms processed in one cycle is equal to 10.

$$\begin{bmatrix} X & X & X \\ X & X & X \\ X & X & X \end{bmatrix} \begin{bmatrix} Y \\ Y \\ Y \end{bmatrix}$$

Nine multipliers can be used to process this vector, matrix product in less than 1 μsec.

$$\begin{bmatrix} X & X & X & X \\ X & X & X & X \\ X & X & X & X \end{bmatrix} \begin{bmatrix} Y \\ Y \\ Y \\ Y \end{bmatrix}$$

Ten multipliers will not process this vector, matrix product in 1 μsec since two terms cannot be processed in this time period.

Upon selection of necessary hardware, the maximum number of units that can be used to compose the machine and still meet the power and circuit size constraints may be computed. At this point the capacity of the machine has been effectively restricted by the constraints and efficiency of operation can be determined. Assume the machine's largest acceptable array configuration is N x N. If a matrix to be processed by this machine is of dimension less than N, the solution is easily obtained, but some processing elements will be idle during the process and still require power. This degradation can be corrected by eliminating the idle units or disabling them until needed (17).

As the dimension of the problem grows larger than the N x N array of the machine, two cases develop. First the computer may be redesigned to

use an array of processors of the fastest available cycle time, or just fast enough to compute the terms of the matrix in the required time with each processor computing several terms.

Array Design Example 2

Specifications:	Data Time--1 μsec System Power (Multipliers)--10 watts Circuit Size--20 square units
Process Hardware: (Specifications)	Cycle Time--450 nsec System Power--1.25 watts Circuit Size--1.75 square units
Array Design Data:	8 Multipliers--10 watts Cycle time less than 1/2 A μsec System Area = (8 mply) (1.75 sq units/mply) = 14 sq units

$$\begin{bmatrix} X & X & X & X \\ X & X & X & X \\ X & X & X & X \\ X & X & X & X \end{bmatrix} \begin{bmatrix} Y \\ Y \\ Y \\ Y \end{bmatrix}$$

8 multipliers can process 16 matrix terms in less than 1 μsec if each multiplier does two terms.

This case deviates from the array processor design criteria, since a one-to-one correspondence of processors to terms is nonexistent. However, as constrained, this design variation is a possible solution to the dilemma. If the array in this machine is composed of N^2 processing units, and the matrix is composed of X^2 terms, then it is easily verified that if I = X/N, where I is an integer, the resulting efficiency of the system will be 100 percent, on the basis of power consumption to work. This relationship is best seen by the following example; if N = 2, then the machine array is composed of 4 processing units. A matrix of dimension 4 x 4 will be composed of 16 terms and the computation process is executed

in 4 cycles of the 4 processing elements with zero idle time. With X and N not related by an integer I, there will exist idle states during the problem execution time. For example, if N = 2 and X = 3, the process will require 3 cycles of each of the 4 computing components, but during the 3rd cycle only 1 processor will be used and 3 will be idle.

The second case exists if slower hardware that is physically smaller and that requires less power is employed in the machine. The propagation of these slower components must likewise surpass the cycle time constraint, but due to its size and power advantages, a larger array is realizable. This scheme will also result in the capability of manipulating a more comprehensive matrix, vector product. The boundary of the second case design exists at the point where the machine dimension equals the matrix dimension.

Array Processor Design Example 3

Design Specifications:	Data Time--1 microsecond System Power (Multipliers)--10 watts Circuit Size--20 square units
Process Specifications:	Cycle Time--900 NS Power Required--0.5 watts Circuit Size--1.0 square units
Array Design Data:	20 Multipliers--10 watts Cycle time less than 1 μsec System Area = 20 square units

$$\begin{bmatrix} X & X & X & X \\ X & X & X & X \\ X & X & X & X \\ X & X & X & X \end{bmatrix} \begin{bmatrix} Y \\ Y \\ Y \\ Y \end{bmatrix}$$

16 slow processors can be used to do the process in less time than 1 μsec.

A final look at the two cases reveals that by utilization of accelerated components, multiple cycles processing can result in improved performance, while retarded cycle time components similarly will result in expanded capabilities. These two cases illustrate some limits to the defined array processor design criteria and imply that efficiency is a function of the relationship of the machine array to the problem array.

Limits on Vector, Matrix Cycle Time

The initial problem placed upon the design consists of an execution time stipulation indicating when the results of the vector, matrix product will be completed. In this specific situation M x N multiplications and N(N-1) additions must be achieved. It has been established that techniques for the reduction of the number of operations of addition and multiplication are nonexistent. The only logical approach capable of improving the overall precipitancy of the calculation is to reduce the time delay of adder and multiplier components. Knowledge of the limitations of addition and multiplication logic will surface to establish the impediments on the size of the problem acceptable to a hardware processor.

Winograd deduced the theoretical lower limits of the multiplier (19) and addition (20) process. In doing so, an (r,d) circuit is utilized to express the terms of the bound of a d-valued logic circuit possessing elemental fan-in of at most r and having the capacity to compute any r argument, d-valued logic function in a unit interval. The addition of two N bit operands approaches lower limitation in the binary number system in compliance with the equation

$$t \geq [\log_r 2N].$$

Winograd further illustrated that the theoretical lower limitation on multiplication delay is consistent with or slightly swifter than the addition limitation equation. The equation is of the form

$$t \geq [\log_r \cdot (N - 2)].$$

Through the correlation of practical circuit delays with the theoretical limitations, insight in practical design can be acquired. The most prevalent technique for the realization of addition is carry-look ahead with a speed exemplified by

$$S = 4[\log_r N].$$

In multiplication, the fastest practical realization implements multiplier encoding techniques combined with a wallace tree interface of carry-save adders and culminates in a speed of

$$S = 2[\log_{3/2} (N)] + 2[\log_r N].$$

A numerical illustration of the performance equations is produced by letting $r = 4$ and $N = 16$ bits. With this assumption the low limits of addition and multiplication can be calculated to be 3 gate delays, whereas the carry-look ahead adder realization possesses 8 gate delays in contrast to 18 gate delays for multiplication as seen in present designs. Existing adder logic more closely approaches the theoretical limitation than does multiplication due to the ideal implementation of addition in the binary number system. The cardinal rule emanating from the investigation of addition and multiplication limitations reveals that a data medium that permits the lower bound of addition will disregard the lower bound of multiplication and vice versa. This postulate is exemplified by the slide rule's capability to compute multiplication with logarithms

while being inept in addition. A similar example is the ROM look-up table calculation media. The ROM system is an inefficient data representation for both multiplication and addition, and in practice it provides comparable access time for both operations.

Hardware Monolithic Multiplier Power and Size Problems

The three most critical features of a multiplier are speed of multiplication, power dissipation, and binary word length. Word length requirements are of function of data accuracy requirements, stipulated by the overall problem (21). In practical applications the number of bits of accuracy implemented should correspond to common existing word lengths of 4, 8, 12, 16, and 24 Bits. The word length indicates the bit length of each operand and half the bit length of the product. If the number of bits required exceeds the common word lengths, then cascaded configurations of multiple multipliers will be interconnected to generate the resultant product. This implies that a fluctuating word length creats a direct alteration upon circuit size parameters in the circuit realization capable of solving a given problem. A secondary concern generated by the word length will result in the need of some knowledge as to the interconnection capabilities of the hardware selected for the implementation of a large word length processor (22). Some monolithic units exhibit adequate speed and power characteristics but require numerous support units to handle expanded word lengths. This will result in an unfavorable circuit size characteristic that violates the circuit area constraint.

The present power dissipation of monolithic mulitpliers ranges from 300 milliwatts on a 2 x 4 multiplier to 5 watts on a 16 x 16 multiplier. At the 5 watt power dissipation level, induction cooling techniques are employed since the wattage is well over practical upper bounds for a monolithic substrate. The thermodynamic equation that indicates the power capabilities of a monolithic circuit is

$$T(\text{junction}) = T(\text{ambient}) + \theta jA(\text{power}).$$

T(junction) is the temperature of the silicone chip, T(ambient) is the still-air ambient temperature, θjA is the thermal resistance of the package, and the last term exemplifies power dissipation. Thermal retardation is expressed as Δ°C per watt; this is stipulated as one watt of heat energy being required to raise the temperature by Δ°C. A typical thermal characteristic of a 40 pin chip is in the range of 30°C per watt. If the ambient temperature, as specified by standard military requirements, is at a maximum of 125°C, and the limit on silicone junctions is 175°C, then the break point on power dissipation is calculated by

$$\text{max power} = (T(\text{junction}) - T(\text{ambient}))/\theta jA$$

$$\text{max power} = (175 - 125)/30 = 1.6 \text{ watts}.$$

This is a textbook solution to power dissipation limitations of a silicone substrate and serves to show that one watt is approximately the maximum power dissipation of an LSI device. To handle power dissipation in the range of 5 watts per substrate, the dip must be constructed with fins or other heat sinks to diminish thermal resistance to around 10°C as shown in Figure 10. LSI manufacturers are constrained primarily by cost and are reluctant to exceed the 1.6 watt power dissipation per chip (23).

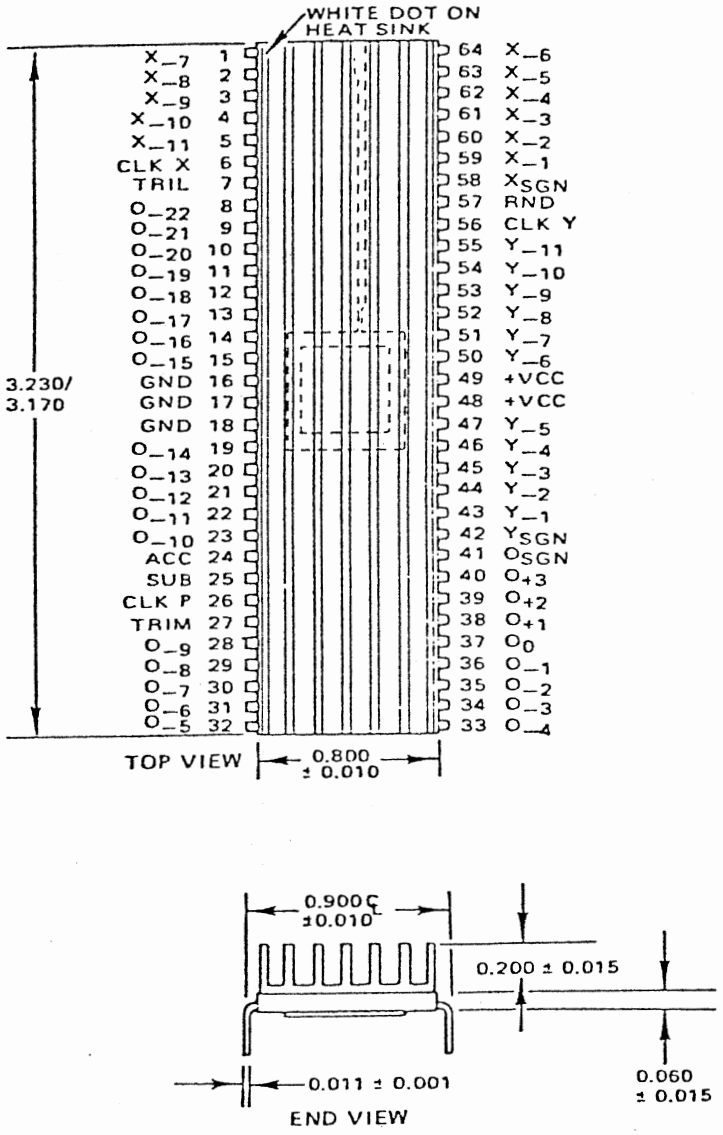


Figure 10. Heat Sink Dip for LSI Device

High-speed multiplication requires special combinatorial algorithms that simultaneously form partial products and add them in one operation. Each sequential bit in the partial product is determined by an AND operation of successive multiplicand bits with a single multiplier bit. This is analogous to the add-and-shift technique, since a zero state bit multiplicand produces a zero partial product and a one state multiplier simply duplicates the multiplicand in the partial product. The equivalent shift operation in a logical multiplier is consummated by the interconnections of the logical adders utilized to sum the partials (24).

A practical procedure for measuring the speed of a multiplication unit is to evaluate the speed as a function of logic-gate propagation delay, while the power dissipation is a function of the total number of gates. Referring to Figure 11, the propagation delay of this 8 bit multiplier algorithm is affected by the delay from A_1 to B_7 to S_{15}. This route is composed of 14 adder units, with 4 gate delays each, with another gate delay for the generation of partial products, resulting in a total of 56 gates. The total gate count is composed of 64 AND gates for the generation of partial products and from 56 binary adders, individually realized from 10 logic gates. The total gate count for an 8 bit combinatorial multiplier system that implements this gate structure is 624 gates (25).

A combinatorial multiplier is significantly faster than a sequential-type system, but vast improvements are necessary to approach the theoretical lower bound of operation. For instance, the carry bits that are transferred between the adders impede the process, and an alternate scheme called carry-look-ahead will account for the bits without addition, and will improve the delay time. Improvement schemes of this nature complicate the structure and increase the gate count as a byproduct to their

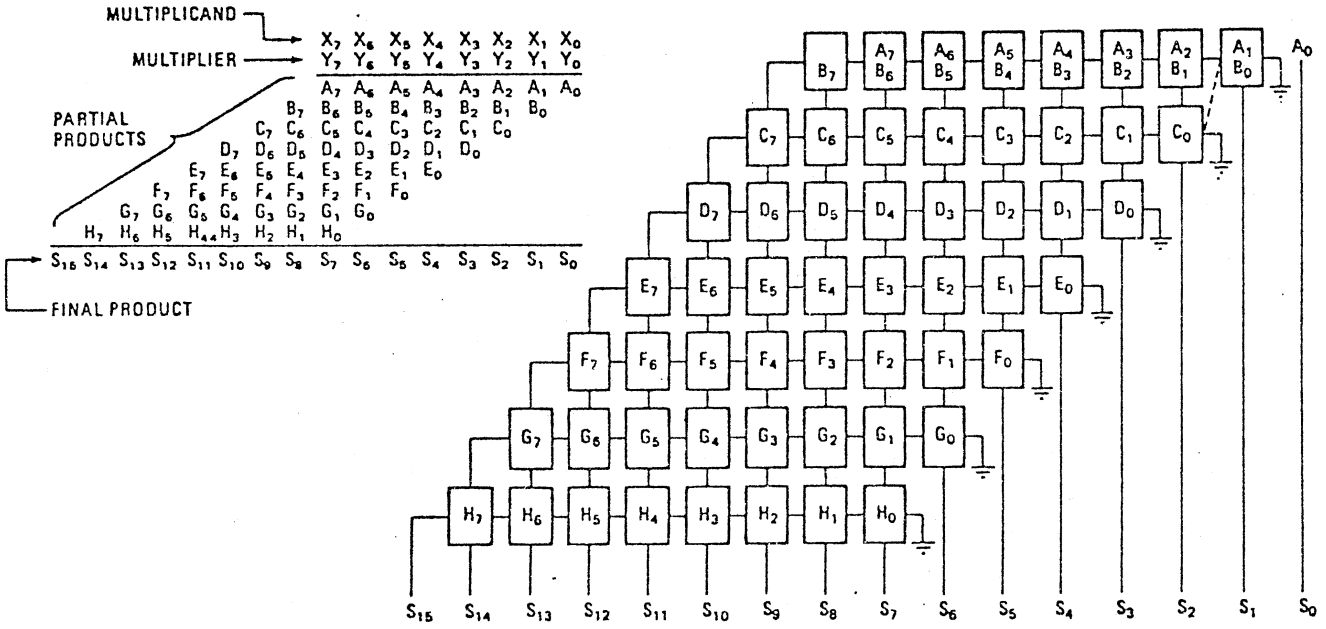


Figure 11. Eight Bit Multiply Algorithm

benefit of increased speed. The larger the gate count per processed bit, the smaller the word length possible per chip, if typical power dissipation limits are upheld.

Techniques for Reducing Multiplier Delays

Alternative techniques have been aimed towards reduction of both gate count and gate delay. One such implementation is designated as the modified Wallace Tree, and succeeds in enhancing the alrogithm by saving all carry bits and adding them in one step using triple input adders. A circuit reduction of 24 gates obtained over the carry-look-ahead scheme by using the modified Wallace Tree. The drawback to both methods is that additional logic is necessary to handle signed numbers. Further research has revealed techniques for accommodating the sign convention and reducing gate complexity through the assistance of encoding practices such as the modified booths algorithm. In implementation, this algorithm requires 675 gates to process an eight multiplier and multiplicand (26).

In the implementation of a multiplication algorithm, several techniques can contribute to a reduction in power dissipation on the substrate. First, computer-aided circuit design studies are employed to determine noncritical data paths. These noncritical routes are acceptably realized with slower gates that dissipate less energy. Second, the number of devices required for AND-OR-INVERT gates in the multiplexer section of the circuit can be optimized. Logic functions in some cases can be realized with single transistors, such as the equation $\bar{C} = \bar{A} \cdot B$. This particular equality is realizable utilizing the collector, base, and emitter of one transistor. A subsequent enhancement scheme is to match

the input threshold voltages of sequential states to reduce the need for translator circuits in the system.

Extensive research into LSI design is invoking higher speed of circuit operation at comparable power levels. These advances will escalate the size of the problem solvable in the same time frame as before, but the power constraints and size problem are constant. This is exemplified by the latest experimental C-MOS and a new bipolar stepped electrode process by Hasashino (Mippon T&T subsidiary) which demonstrates a 0.5 psec propagation delay with a 0.1 pJ power delay product. This repercussion constitutes an order of magnitude improvement in propagation delay over D-MOS and V-MOS, while still maintaining the power delay product (27).

It has been implied that the end of the bounding improvements for photo-mark-generated LSI circuits will occur as the point line width approaches the visible light wave length. Before LSI densities can mature further, techniques must be conceived which allows a line width reduction to below 1 micron. Electron beam lithography (EBL) exemplifies such a capacity by projecting integrated circuit patterns directly, without the aid of masks and contact printing of the substrate. Recent advances in EBL have led to the concept that this technology will mature rapidly within the next few years and overtake the present problems, particularly the constraint of high cost. EBL-generated transistors will exhibit lower power densities, due to their smaller physical size. If gate complexity is escalated tenfold and chip size by 4, the resulting dimension of the substrate will reach 12mm x 12mm, and the one million elements per chip level can be approached. The culmination to the realizable number of devices per chip will ultimately be limited by power dissipation and the number of input and output lines required for

suitable applications. Generally speaking, as the logic function implemented on a chip becomes more complex, it becomes more specialized, fewer total devices are utilized, development costs inflate, and the task becomes uneconomical (28).

The conclusion reached by an analysis of monolithic multipliers can be cataloged as follows:

1. The number of gates necessary to implement a multiplier algorithm are semi-constant from one algorithm to the next.

2. If the gate requirements to implement an 8 bit multiplier unit are assumed in the range of 650 gates, then primary consideration as to the power and speed of the unit is the type circuit technology utilized.

3. It can be stipulated that to increase speed, an increase in power dissipation is required.

4. Regardless of recent innovations in technology, power, size, and cost constraints are still applicable.

Multiple Phase Processing

As the technology develops and processors become less expensive, faster, and less power-consuming, the design constraints of speed, power, and circuit size will still exist. This has been evident throughout the transition from tubes to transistors and transistors to LSI circuits. With all the innovations in speed, power, and size over the last several decades, problem sizes have increased, requiring further consideration to the speed, power, and size dilemma.

In the realm of vector, matrix operations, the effects caused by design constraints are functions of the problem's characteristics. If a 3 x 3 matrix and a 3 x 1 vector are multiplied together, 9 multiplication

operations and 6 additions must be performed during the time prior to the result being made available. The cycle time of a processor is the time necessary to complete one computation, and the completion time is the total time needed to complete the solution. The utilization of four processors to compute the product of a 3 x 3 matrix and a 3 x 1 vector will result in three cycle time intervals of the processors, assuming that the cycle time is one-third or less of the total computation time. In the interim of the first cycle, four terms are computed and retained; during the second cycle, four more terms are evaluated. The last cycle will contribute only one term to the partial terms necessary to complete the solution. During the closing iteration, all processors will be operating but only one will be doing useful work.

An alternate scheme for evaluation of this vector, matrix product will be to bring into operation two or more processors of dissimilar cycle times and by so doing, alter the time, power, and size variables. Such a processing unit will be designated as a multiple phased array processor. This design will conform to the array processor design criteria as each unit that is affiliated with a particular cycle time will be executing the same instructions concurrently on different data streams. This process is equivalent to operating several array processors of different speeds in parallel to improve the performance of the computer and meet the constraints of time, power, and size. Adopting this multiple phased array concept in conjunction with the knowledge that power is directly related to speed and inversely related to size, invokes a situation to which optimization techniques can be applied. In the discussion of the matrix, vector multiplication problem using a matrix of dimension 3 x 3, a solution could have been obtained in the required time frame by

computing four terms with one fast processor concurrent with the computation of three terms using a slower processor unit. The remaining two terms are evaluated in the same time frame by still another processor of even slower speed that is capable of only two cycles in the time required for the fastest processor to perform four complete operations. Through the utilization of such a scheme of computing, the design problem can be solved such that the cost is minimized and constrained by time, power, and circuit size.

Optimal Design With Linear Programming

A linear program is defined as a mathematical model which is designed to obtain a set of nonnegative numbers or variables which maximize or minimize a linear equation or object function while satisfying a system of linear constraints. It is apparent in this situation that the linear formulation must consist of and result in an integer solution.

Utilizing matrix notation, an integer program is exemplified as follows:

Minimize:

$$C_1P_1 + C_2P_2 + \ldots + C_NP_N = Z \tag{4.1}$$

Subject to:

$$A_1P_1 + A_2P_2 + \ldots + A_NP_N \leq B \tag{4.2}$$

P_i is an integer, $i = 1, 2, 3, \ldots$

C_i, $i = 1, 2, 3, \ldots$ is a cost term

A_i, $i = 1, 2, 3, \ldots$ is an $N \times 1$ vector

B_i, $i = 1, 2, 3, \ldots$ is an $N \times 1$ vector of constraints (or simply, the right-hand side)

P_i, i = 1, 2, 3, . . . is an N vector of integers.

In this application the objective function (Equation (4.1)) is the total cost of the array of processors used by the computer to process the matrix. Equation (4.2) is composed of three or more constraint equations.

To evaluate the optimal design of the multiphase array processor, certain data on each processor must be obtained.

1. Cost of each type of processor considered.
2. Time necessary to complete one computation.
3. Power (in watts) used to operate each type processor.
4. Number of packages that compose each processor and number of pins used on each package.

Once the hardware is acquired, the linear program equations are subsequently created. Let C_i = cost of processor P_i, i = 1, 2, 3, . . .

$$C_1P_1 + C_2P_2 + C_3P_3 + \ldots + C_NP_N \leq Z$$

$$C_i \geq C_{i+1}, \quad i = 1, 2, 3, \ldots N.$$

Let T_c equal total time allowed for matrix computations and T_{pi} equal the cycle time of each processor P_i,

$$T_i = \text{largest integer } (T_c/T_{pi}), \quad i = 1, 2, 3, \ldots$$

The time equation will be in the following form:

$$T_1P_1 + T_2P_2 + T_3P_3 + \ldots + T_NP_N \geq (\text{number of terms in matrix}).$$

Let P_i equal the aggregate of processors of type P_i necessary to compute the problem if only type P_i processors are employed.

$$P' = (\text{number of elements in matrix})/T_1.$$

Let W_i equal the power in watts necessary to operate each unit P_i. The power constraint equation is formed as follows:

$$W_1P_1 + W_2P_2 + W_3P_3 + \ldots + W_NP_N \leq W_T,$$

where W_T represents the cumulative power sanctioned for consumption by the array hardware. W_T possesses an upper and lower bound that are computed using P_i'. The upper limit is obtained from the largest term of the limit equation

$$W_iP_i' = W_i', \; i = 1, 2, 3, \ldots$$

The lower limit is the smallest value W_i', as acquired from the calculations. These upper and lower boundaries will bracket the possible power range that encompasses the choice of W_T.

The area equation is attainable by scaling each processor as to the quantity of square units it requires on the circuit board with allowance made for the bus structure, power, and circuit board configuration. Let U_i equal the necessary units for implementation of processor P_i. The subsequent equation will be

$$U_1P_1 + U_2P_2 + U_3P_3 + \ldots + U_NP_N \leq U_T.$$

The right-hand side, U_T, is the total circuit realistate allocated for the array structure hardware and bus system. The limits on the range of U_T are obtained from the W_i' terms generated from the equation

$$W_i' = U_iP_i, \; i = 1, 2, 3, \ldots$$

The largest and smallest terms of W_i' provide the upper and lower limits for the value of U_T.

The resulting linear program is of the form:

Minimize:

$$C_1P_1 + C_2P_2 + C_3P_3 + \ldots + C_NP_N \leq Z$$

Constraints:

$$T_1P_1 + T_2P_2 + T_3P_3 + \ldots + T_NP_N \geq \text{Size}$$

$$W_1P_1 + W_2P_2 + W_3P_3 + \ldots + W_NP_N \leq W_T$$

$$U_1P_1 + U_2P_2 + U_3P_3 + \ldots + U_NP_N \leq U_T.$$

The solution to the linear program will exist in a region bounded above the time line and below the power and area lines. Prior to attempting to obtain the optimal solution, the solution region should be examined to determine if it exists in such a state that will allow the existence of a feasible solution. At this point a reduction or increase of the solution region is achieved by altering the values of W_T and U_T. This capability will facilitate the search for the integer linear program solution by effectively reducing the search domain.

The solution to the integer linear program is generated by using available computer software and computer systems. The technique is to use a branch and bound algorithm based on the Land and Doig (32) method. Details of the algorithm are covered in Appendix A. The end result of the linear program will be a circuit of a practical nature in an optimal form to solve a vector, matrix product computation.

Optimal Two Phase Design

A graphic illustration of the parameter characteristics of the design scheme is easily viewed in a two-dimensional problem. Assume the

matrix to be processed has 30 terms arranged in a 5 x 6 array. The variable size will be equated with the number of terms composing the array.

$$\text{Size} = 30.$$

The total processing time for the array is set at one μseconds.

$$T_c = 1 \text{ μsec}$$

This interpretation of T_c implies that the entire array will be processed in 1 microsecond or less, with the system ready to undertake the next operation. From the stipulations on T_c, the selection of adequate hardware can be resolved. First, the cycle times of each of the two types of processors must fall below the T_c value. Let the individual cycle times of the two types of processor be specified as T_{p1} and T_{p2}, which results in time constraint parameters

$$T_1 = \text{largest integer } (T_c/T_{p1})$$

$$T_2 = \text{largest integer } (T_c/T_{p2}).$$

The consequence of the parameter values is depicted in the time constraint equation

$$T_1P_1 + T_2P_2 \geq \text{Size}.$$

The boundaries of the equation acquired in the form of P_1' and P_2' are

$$P_1' = \text{Size}/T_1$$

$$P_2' = \text{Size}/T_2.$$

The variables P_1' and P_2' each represent the quantity of processor units essential to compute the solution if only units of type P_1 or P_2 are employed in the design. P_1' and P_2' subsequently introduce the limits on

the maximum number of computing components fundamental to the problem.

The power specification of processor P_1 and P_2 are, respectively, W_1 and W_2. This results in the subsequent power equation

$$W_1P_1 + W_2P_2 \leq W_T.$$

The limits on the power equation are generated by alternately zeroing P_1, then P_2, and in each case computing the number of units necessary to handle the problem. The limits are evaluated in terms of

$$W_1' = W_1P_1'$$

$$W_2' = W_2P_2.$$

The larger of the two values W_1 or W_2 represents the upper limit and the other equates to the lower limit. Between these limits the design value of W_T is obtained. The boundaries stipulate the practical range of W_T values that can be employed in a realizable design.

The circuit area constraint equation requires more in-depth consideration prior to its evaluation. The appraisal of the circuit area is a function of the circuit board construction technique employed, as well as the bus structure utilized. The numerical quantities U_1 and U_2 are indicative of the area required to fabricate one hardware unit of type P_1 and one of type P_2. A prime consideration of the fabrication technique should be the cost function connected with the production of the circuit. Once the formality of the circuit area requirements is determined, the area equation is generated in the form:

$$U_1P_1 + U_2P_2 = U_T.$$

With the use of P_1' and P_2', the limit constraints on the equation are procured in a similar fashion as those of the power equation.

$$U_1' = U_1'P_1'$$

$$U_2' = U_2'P_2'$$

The values of U_1' and U_2' will produce the boundaries bracketing the selection of variable U_T. The resulting problem equation is written in the form:

Minimize:

$$C_1P_1 + C_2P_2 \leq Z$$

Constraints:

$$T_1P_1 + T_2P_2 \geq \text{Size}$$

$$W_1P_1 + W_2P_2 \leq W_T$$

$$U_1P_1 + U_2P_2 \leq U_T.$$

Assume processors of type P_1 are capable of two cycles in the time period T_c, while processors of type two can execute only one cycle. Also allow the power and unit area needs of both processors to be equal. This will impose constraint equations of the form:

$$2P_1 + 1P_2 \geq 30$$

$$1P_1 + 1P_2 = x,\ 15 \leq x \leq 30$$

$$1P_1 + 1P_2 = y,\ 15 \leq y \leq 30.$$

Figure 12 shows the time equation plotted in contrast to the upper and lower limits of the power equation and Figure 13 shows the time line and limits of the area equation. Different constraints on both power and area will produce lines on the plot parallel to the boundary lines, such as the lines shown in Figure 14. With the capability of moving the power

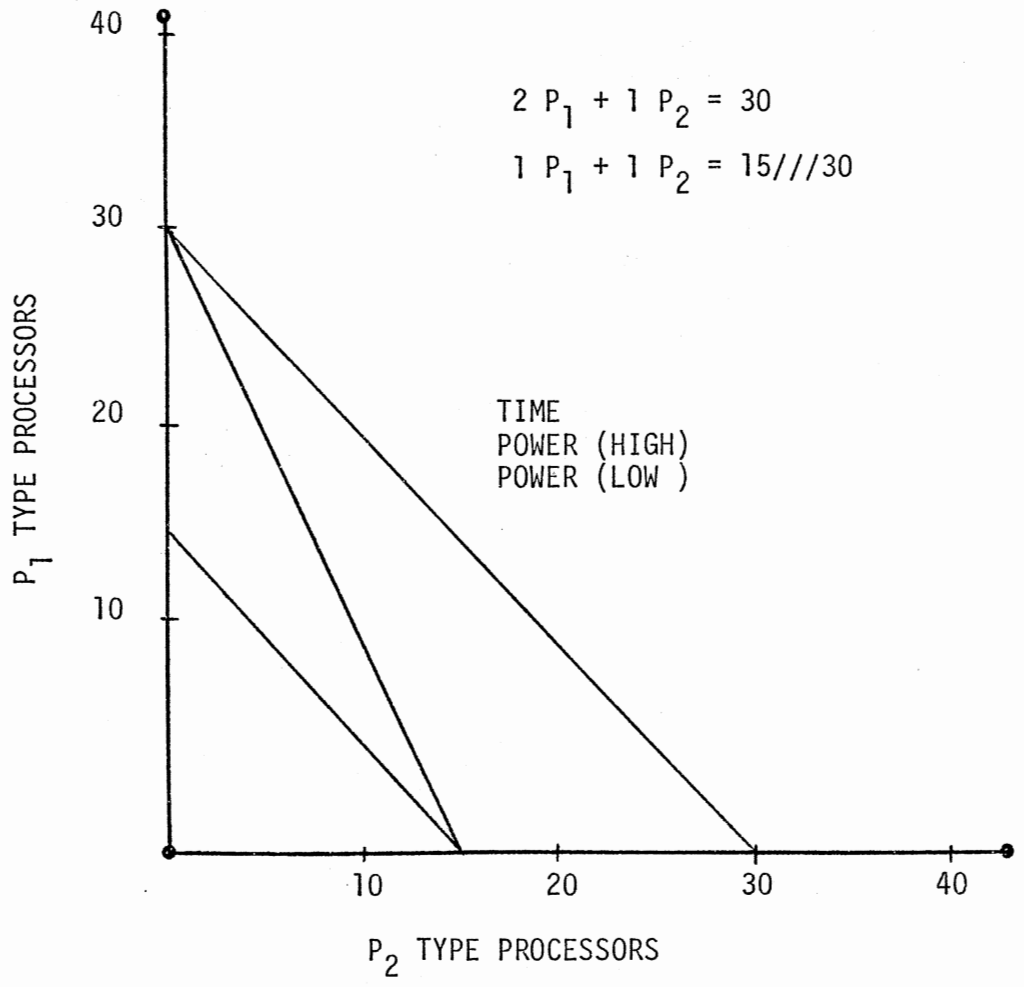


Figure 12. Time and Power Equations

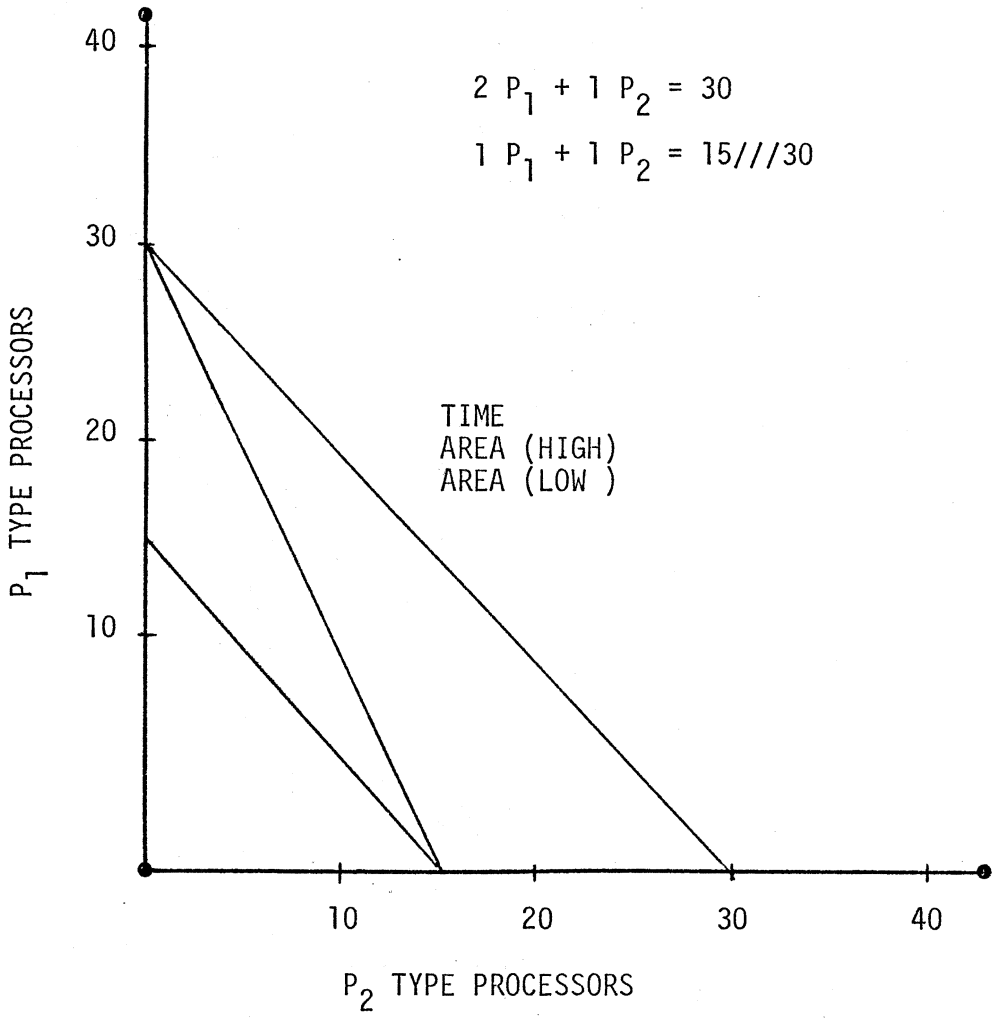


Figure 13. Time and Area Equations

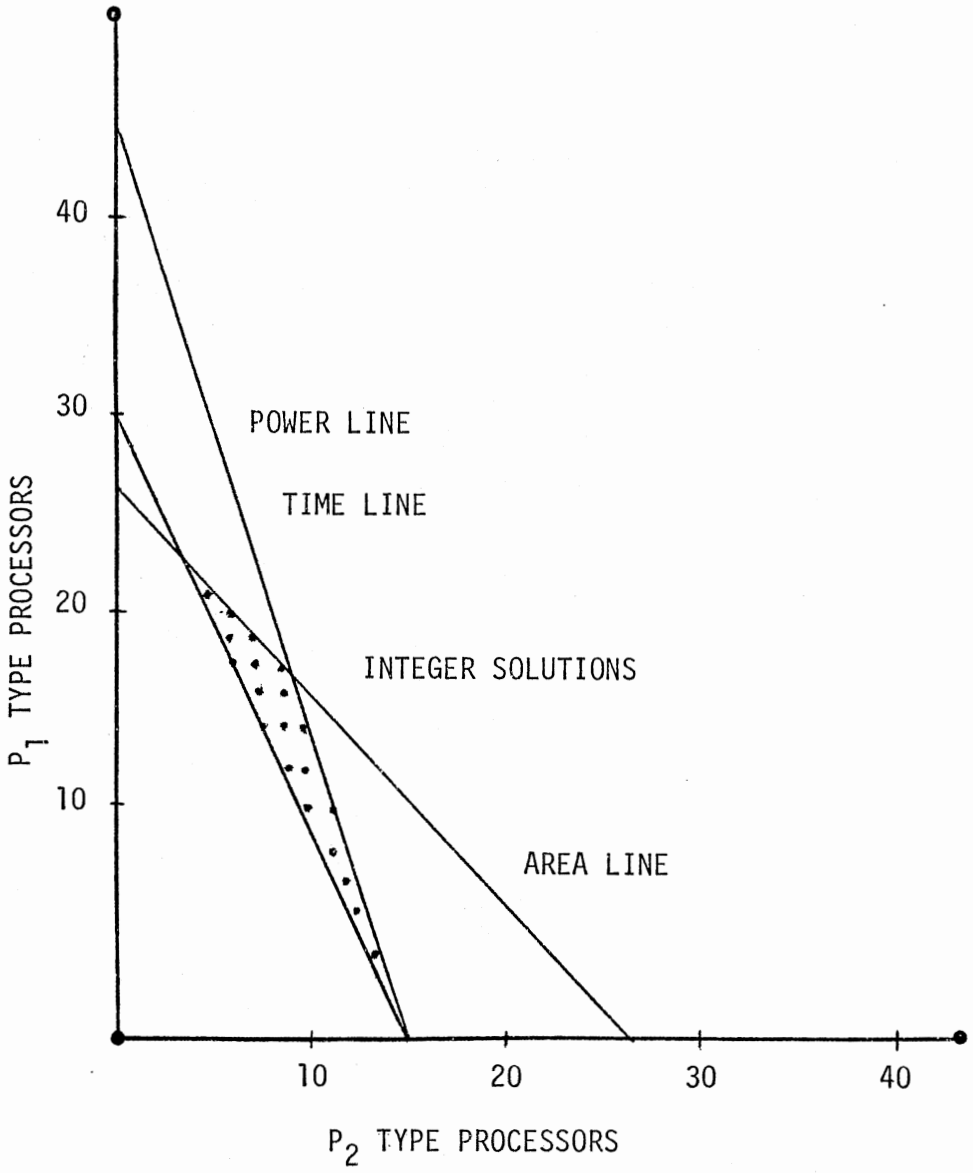


Figure 14. Integer Solutions of Problem

and area equation lines on the plot, it is possible to reduce the region in which the solution will reside. Figure 14 illustrates a linear program plot with the time, power, and area equations shown. The integer solutions are also shown on the plot as they occur in the region above the time line and below the power and area lines. The optimal solution is obtained by testing each of the solutions in the cost equation and determining which solution gives the minimum cost to the system. The optimal solution can be computed either graphically for small problems or by using the computer program discussed in Appendix A. Prior to the use of the optimal program, a program of the type shown in Appendix B can be used to plot the linear equations or just check the boundaries of the constraint equations to assure that it is possible to obtain a solution with the constraint values.

Summary

The design characteristics of an array processor can be altered to produce a multi-phase form, capable of optimal operation. The concept is to utilize array processor components of different cycle times and effectively operate them in parallel in a SIMD environment. Using the operating characteristics of available hardware, a designer can formulate a set of linear equations, solvable with standard linear programming software, and generate a practical circuit configuration. The entire design package lends itself well to an interactive computer program. Software of this nature would complement a designer's ability to make decisions as to practical design capabilities of available hardware. A summary of the steps used to design the Multi-phase processor system and a sequence flow chart are given in Appendix E.

CHAPTER V

PROCESSING ELEMENT DESIGN

Introduction

Solution of the optimal linear program marks the culmination of the overall circuit configuration problem. Subsequent design problems are approached ultilizing similarly defined constraints imposed on the overall system. The SIMD definition stipulates that individual processors must simultaneously operate on different data streams. In addition, the array processor subclass definition further invoke that all processing elements are to perform identical operations concurrently on different data streams under control of one instruction. Subsequently, high speed data paths are imperative between processor units. The cardinal directive of the subclass is that all data fetched in one time frame is processed in one time frame and stored in one time frame. These and other stipulations that categorize array processor design must translate into design criteria of the individual processor units that compose the array system. A relationship between the structure exhibited by the problem and the concept of the hardware design should exist analogous to the constraints placed on the origonal design concept.

The Design of Processor Bus Structure For Vector, Matrix Products

If the hardware structure internal to the processing unit is based

on the algorithm used, a study should be conducted to coordinate certain steps in the algorithm to the implementation of hardware. Initially, determination of the sequence of the vector operation (multiplies and additions) should be obtained in an effort to postulate the maximum number of concurrent operations that can co-exist in one time interval. The operations fundamental to an N by (N+M) matrix will be the order of N (N+M-1) additions. The dilemma is to determine in what order the multiplications and additions should occur in order to maximize the array capabilities of the system.

Certain ad hoc methods of studying the possible combinations of multiplications and addition schemes in a matrix, vector product will provide insight into the complexity of a combined algorithm hardware solution. These methods culminate in the realization that an overall analytical procedure is imperative to deduce a practical solution. A technique suggested by Torng and Wilhelm (29) to optimize interconnections of central processor registers suggests that the maximum number of data lines is best determined by using linear dynamic programming methods. The Torng and Wilhelm technique is initiated by defining a transfer matrix consisting of P x P elements, where each term of the matrix represents a transfer from one register to another. The matrix assists in charting concurrent and sequential operations between registers in a computer in order to determine the minimum bus requirements and maximum data flow. This concept is expandable to an array system to maximize data flow and minimize the bus structure.

To utilize this technique on a distributed architecture system that is capable of solving a vector, matrix product requires the formation of two transfer matrices. The first matrix (Figure 15) is established to

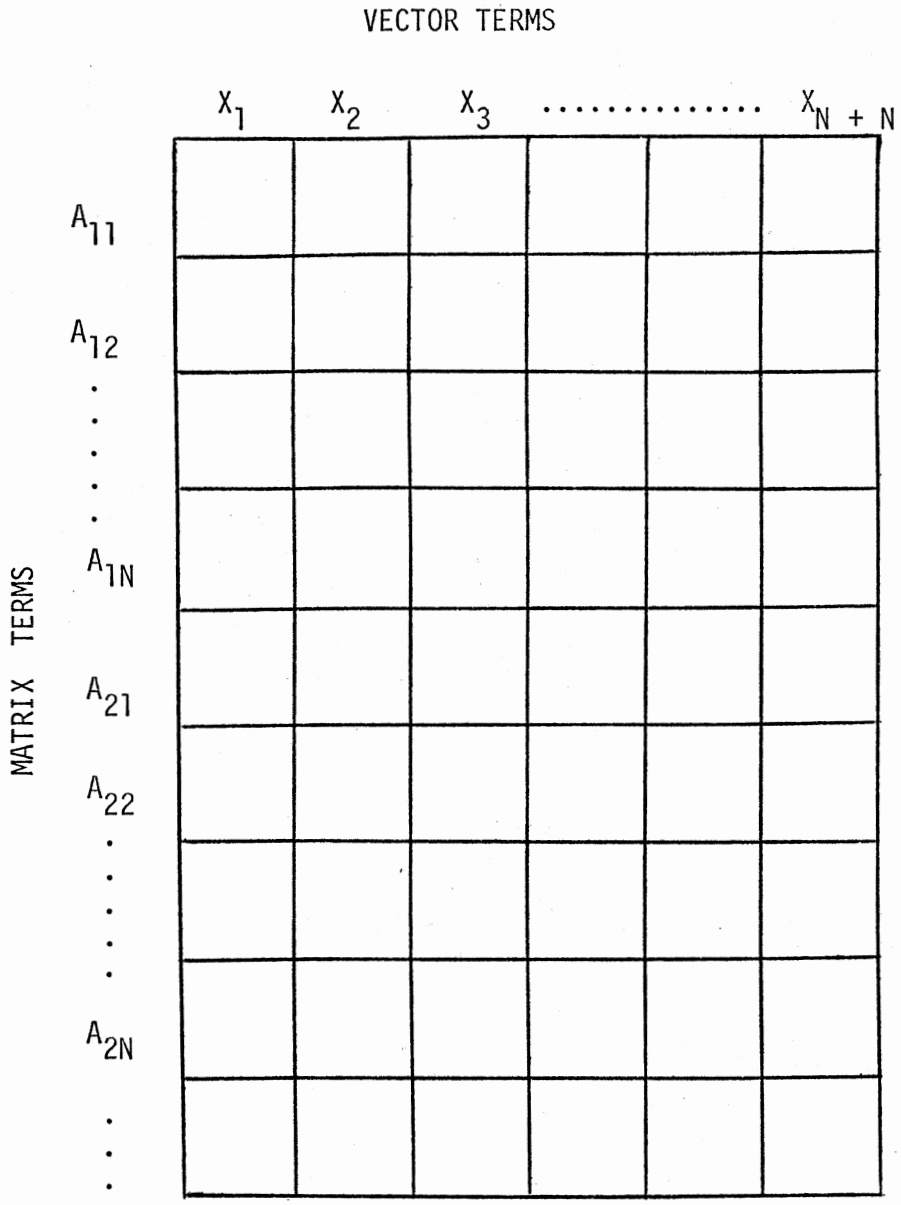


Figure 15. Multiply Table

study the addition schemes. This matrix is composed of columns designated by the terms in the vector of the vector, matrix problem. If the order of the problem matrix is N by (N+M), then the first transfer matrix will have N + M columns and N (N+M) rows. A check mark placed at the intersection of a column and a row designates a required multiplication. This table is constructed first since the multiplication must precede the addition in the solution of the product.

Under the SIMD array processing criteria, all data values that are processed in one interval must be fetched in one interval. It is evident from the multiply table that if (N+M) elements of the vector are fetched in one period, the N (N+M) multiplies could be accomplished in one time interval. This is evident by scanning down the columns and noting the total number of checks in each column that correspond to elements that have been fetched. Note, that to fetch the (N+M) vector elements in one interval requires (N+M) bus systems to provide data transport from their storage locations. Furthermore, to accomplish the parallel processing of the N (N+M) multipliers and N (N + M) additional bus paths to fetch the elements of the matrix in one time interval. An examination of rows of the multiply table shows that fetching any one element of the matrix will result in only one possible operation. However, by looking down the columns, it is evident that if one vector element is fetched, then N multiplies are concurrently realizable by fetching the one vector element and N matrix elements.

The formation of the addition table is somewhat less routine as the terms must be separated into two groups of relatively equal size and deposited on the extremities of the table with one group of terms on the top and one group along the side. Check marks are placed on the grid

corresponding to terms that must be added, with the stipulation that addition of a term occurs only once on the chart. From this matrix grid, it is evident that only VAL number of additions may occur concurrently, where

$$VAL = \text{largest integer } [N\ (N+M)/2].$$

There are N (N+M-1) additions necessary with (N+M-1) addition terms occurring in each row that must take place in pairs. Therefore, [VAL1] designates the number of additions possible in one row in a single time interval.

$$[VAL1] = \text{largest integer } [N\ (N+M-1)/2].$$

The remainder (R) term of this integer division indicates an additional adder interval requirement. From Figure 15, note that if column one of the matrix is fetched, it will require N+1 bus paths, and the N products consisting of terms produced by multiplying X_1 times each of the column terms are generated with N multipliers. The N partial terms are concurrently stored and then N more are produced and added to the stored terms and so on across the row. This action constitutes a maximum parallel operation of multiplication and addition with a minimum hardware requirement. This process reduces the need of tree configured circuits that reduce the parallel processing capability of the system. This result is in agreement with Torng and Wilhelm (29), since they have pointed out that the number of bus paths necessary is equal to the number of simultaneous data transfers required.

This structure will require a multiply, add, and accumulate technique for each element to be processed in parallel and N+1 bus paths used to fetch data. Note that if the multiplier is fed from a magazine loader or

first-in-first-out buffer, that the control and bus path complexity is significantly reduced. To adhere to the requirement of array processing, the N results accumulated in the scheme will require N bus paths to return the data to memory concurrently as shown in Figure 16. This scheme implements N + 1 bus paths and the capabiility of storing the N new values into memory synchronously. The memory realization is possible by utilizing shift registers as the memory for the Z's and Y'z, since they need to shift one value of Z_i onto the input bus in a sequential action. This action will open the required area in memory to allow the accumulated terms to be stored at the completion of a cycle. The A and K memories will operate in a similar manner and can be implemented with a shift register system of memories.

Design of Internal Processor Configuration

The completion of the processor bus structure leads to the design of the internal interconnections of the individual processor units. The original design criteria carry over into the design of the internal hardware of the processing elements. The concept is to maximize the throughput in the processor unit by implementing the maximum parallel operation and data flow.

Assuming the use of floating point two's complement numbers, it is necessary to first define the unit necessary to perform a multiply, add, and accumulate process on two signed binary numbers. The mantissas must be multiplied and the exponenets must be added to produce a floating point product. This product must be shifted to the right or to the left, and its exponent increased or decreased for each shift to make the exponent match the exponent of the accumulated sum to which it will be added.

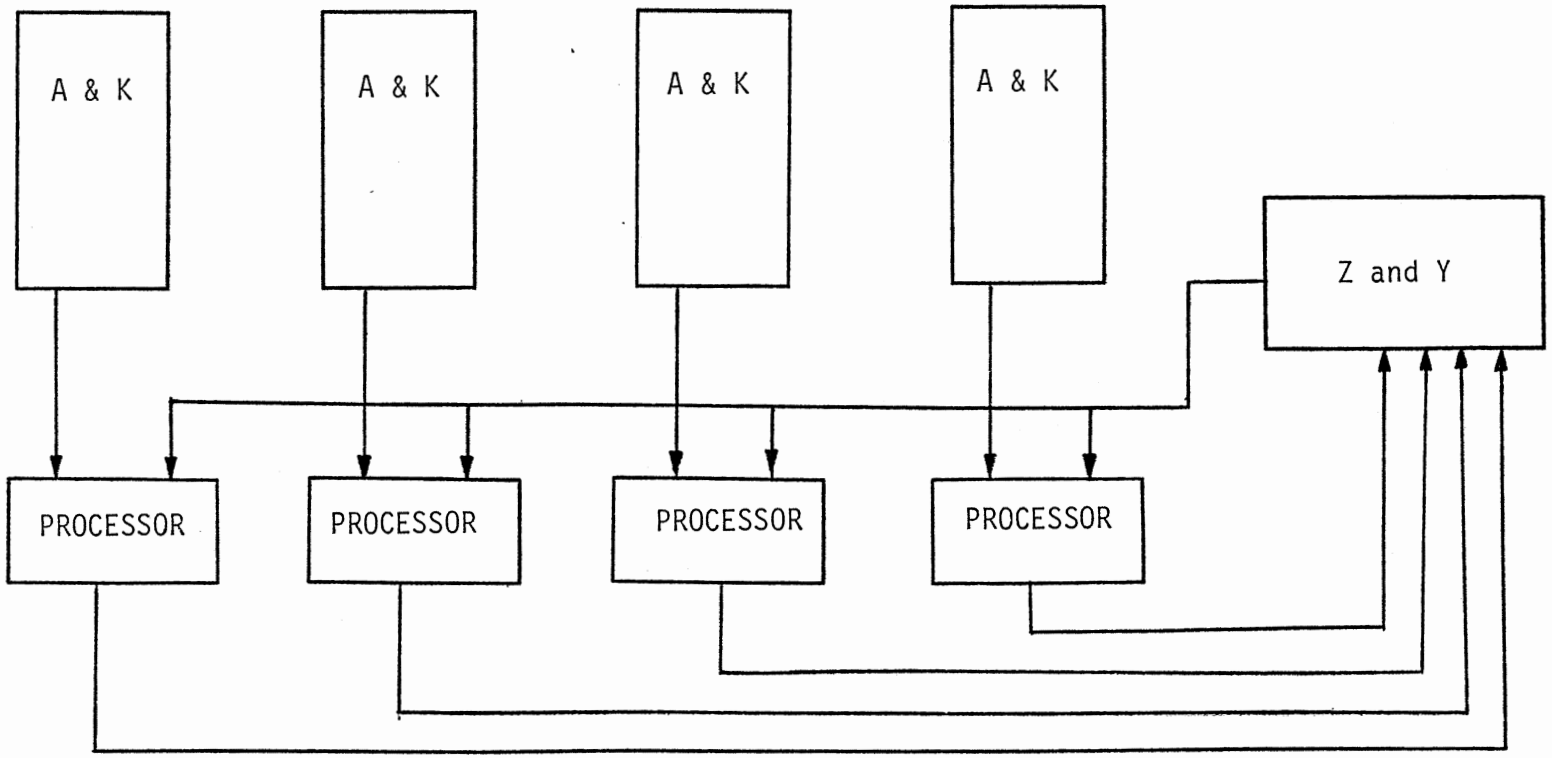


Figure 16. Circuit for the Parallel Storage of Data with Least Bus Lines

The standard structure to perform these operations is of the form shown in Figure 17.

Assuming the elements of the matrix are made available in normalized two's complement floating point format, prior to the calculations, a great saving in design complexity can be obtained. The knowledge that all numbers will arrive at the processor input in the same format will allow the renormalization and shifting of exponents to be accomplished in parallel with the multiplication and thereby increase the throughput by not waiting until the product is formed to check its exponent and shift the mantissa as required prior to addition. For example, a 32-bit floating point number will have a 23-bit mantissa plus one sign bit and a 7-bit exponent plus a sign bit. The multiplication of two 23-bit numbers will typically require 250 ns, and the addition of two 8-bit numbers will require around 60 ns with present technology. Once the exponent of a new product is formed, it is necessary to compare it to the accumulated exponent to determine the number of right or left shifts necessary prior to adding the new product to the accumulated sum. This will be done by subtracting the new exponent and accumulated exponent and will require typically another 70 ns. At this point, the number of shifts of the mantissa necessary prior to addition to the accumulated total is known and approximately 120 ns remains before the next mantissa becomes available. Since the numbers going into the multipliers are both nornalized, the results of the multiplications will require at most one left shift to place it in normalized form (17). It is possible to determine if the product will require shifting after multiplication by analyzing the multiplier and multiplicand prior to the multiplication. Given prior knowledge of the resulting position of the product, in terms of normalization, will allow

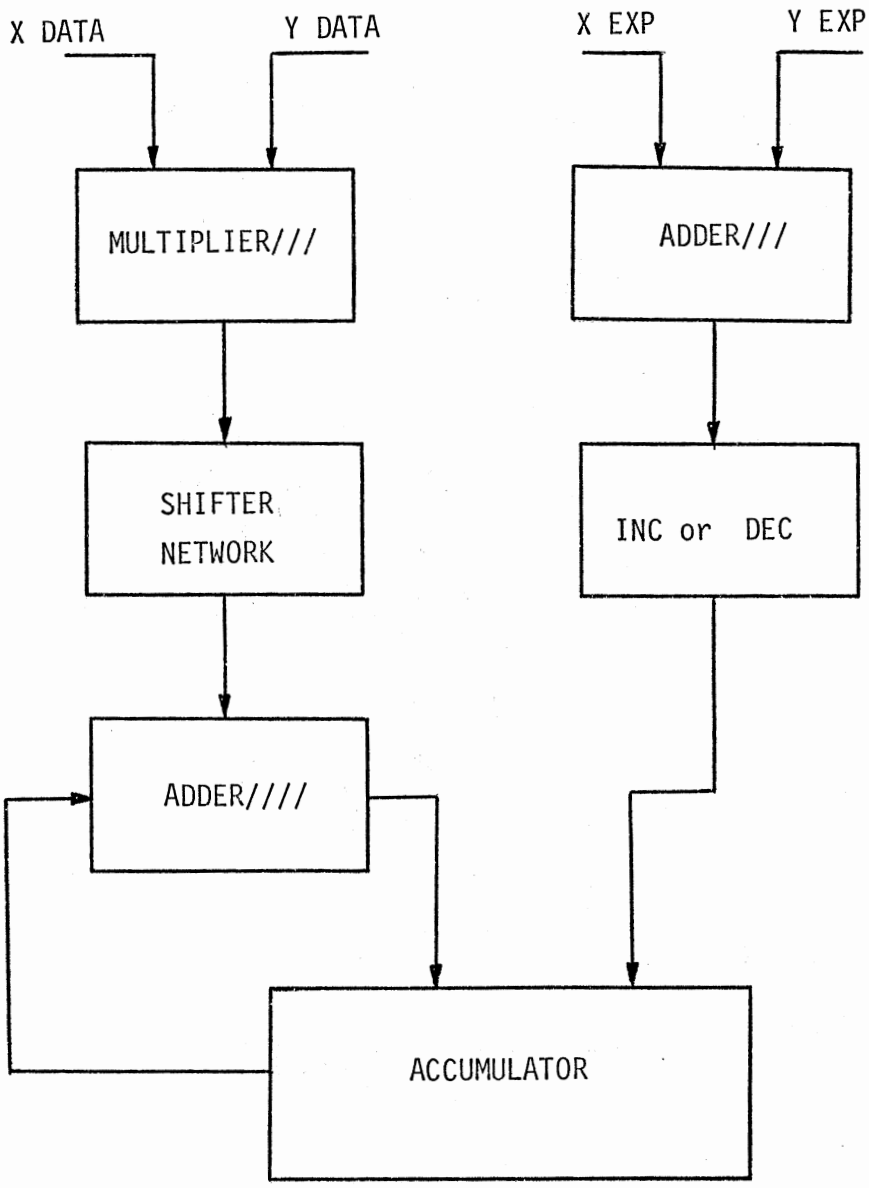


Figure 17. Floating Point Multiplier

the 120 ns remaining before the products are available can be used to set a shift network of multiplexers such that the output of the multiplier may be fed into the adder without further delay. This will preclude the normal procedure of latching the product, and checking and shifting it, prior to the add and accumulate process. A similar method must be used to place the accumulated sum into normalized form prior to storing it in memory for use in later computations. It is possible to process the accumulated data in the last adder stage with the use of look-ahead-carry techniques which will adjust a shift network that places the data on the bus in normalized form.

Two problems remain to be faced in the processing units. First, the size of the accumulator to hold the row products must be determined so that an overflow of the accumulator will not occur. Second, the accumulated element products must be rounded to the proper number of bits to match the data bus and memory size.

The maximum size of any product term $A \cdot X_i$ is

$$2^{\circ} + \sum_{K=0}^{N-2} 2^{2N-1-K} \quad ,$$

and there are $N+M$ products in a row to be accumulated. N equals the number of bits in the multiplier or multiplicand, as well as to the size of the data bus.

The maximum size of the accumulator is then

$$S = (N+M)(2^{\circ} + \sum_{K=0}^{N-2} 2^{2N-1-K}) .$$

The number of bits in the second term will be 2N (where N = the number of bits in one data word). Therefore, the required number of bits of the

accumulator will be 2N plus the number of bits necessary to express the term (N+M) in binary. For example, if there are 8 bits in a data word, then there will be a maximum of 16 bits in the product, and if there are 5 elements in the matrix row, then (5 = 101) or 3 bits will be necessary and maximum size of the accumulator will be 19 bits. The need for shifting the accumulator after each addition of a new partial product can be stopped by allowing enough bits in the accumulator register to stop overflow under the worst case conditions. Once the terms are added in the accumulator, they may be rounded after they are normalized. There are two possible normalization methods commonly used: one is to make the lowest bit of the bits to be kept a one; the other is to add the MSB of the bits to be discarded to the bits to be retained. The fastest method to process the data will be to carry guard bits to allow only truncation or rounding at the interface to some other device which is driven by the output of the computations.

Look-Ahead Shift to Increase Floating Point Operations

Assuming that the input numbers to the floating point multiplier are in two's complement normalized form, the product will appear as

$$S_x . \; X_n \; X_{n-1} \; X_{n-2} \; \ldots \; X_o \quad \text{Multiplicand}$$

$$S_y . \; Y_n \; Y_{n-1} \; Y_{n-2} \; \ldots \; Y_o \quad \text{Multiplier.}$$

By definition of normalization, X_n and Y_n will be equal to one and S_x and S_y will be equal to zero for positive numbers. The largest product possible from an N Bit Multiplicand and an N Bit Multiplier will be

$$\text{Value max} = 2^{\circ} + \sum_{K=0}^{N-2} 2^{2N-1-K} .$$

The form of this product is best illustrated by examples.

11	111	1111	11111
x 11	x 111	x 1111	x 11111
1001	110001	11100001	1111000001

The 2° bit is always set as well as the higher order N - 1 bits, where N is the number of bits in the multiplier or multiplicand. Note that these examples result in positive normalized products and do not require a left shift in any of the cases. Furthermore, as long as the numbers are normalized prior to multiplication, they will at most only require one shift to the left as a consequence of the normalization, which is established when the MSB and the sign bit are not equal. This normalization requirement is always true except for the binary numbers in the form: S.XXX = 1.1000...

The question is how can knowledge of the shift or no shift situation be obtained prior to the multiplication operation. There are four cases that must be studied. Case 1 will consist of both X and Y being positive normalized numbers with X_1 and Y_1 equal to one by definition of normalization.

$$X = S.\ X_1\ X_2\ X_3\ \ldots\ X_N$$

$$Y = S.\ Y_1\ Y_2\ Y_3\ \ldots\ Y_N$$

If X_2 and Y_2 are both 1 and all lesser bits are zero, the product will be of the form $2^{2N-2} + 2^{2N-4}$. This result will be the smallest product value resulting from the multiplication of two N bit numbers having the two MSB of each number set to one's. The results show that a product of

two positive normal binary numbers with X_1, X_2, Y_1, Y_2 equal to one will never need a shift in the product to normalize the result.

For Case 2, let $X_1 = Y_1 = 1$ by definition of normalization and let X_2 and $Y_2 = 0$. Since the largest product possible under these circumstances will be of the form

$$\begin{array}{l} \text{S.1 0 1 1 1 ...} \\ \times\ \underline{\text{S.1 0 1 1 1 ...}} \end{array} ,$$

it is well known that the product of two N bit numbers will produce at most a 2N bit product. Also the product of 2N bit numbers (where N = 4), which are in the form states in this case, will result in a 2N bit product that will require normalization. A method must be determined to obtain the shift information from the multiplier and multiplicand in this case. This is accomplished by holding the multiplicand in the form

$$\text{S.1 0 1 1 1 ... } X_N$$

and finding a multiplier of the form

$$\text{S.1 0 } Y_3\ Y_4 \text{ ... } Y_N$$

that will just cause an overflow into the 2^{2N} bit position. The difference between these two numbers will be the range that must be tested by addition in the limiting case. For example, two 6-bit numbers of the form S.1 0 1 1 1.

$$\begin{array}{l} \text{S.1 0 1 1 1 1} \\ \times\ \underline{\text{S.1 0 1 1 1 1}} \end{array} = \begin{array}{r} 47 \\ \times\ 47 \\ \hline 2209 \end{array} .$$

These two 6-bit numbers will produce at most a 12-bit product.

Bits	12	11	10	9	8	7	6	5	4	3	2	1
Max Value	2048	1024	512	256	128	64	32	16	8	4	2	1

Note that if no shift is required, the product will exceed 2048 (12th bit position value). To find the shift value, the product must be less than 2048. By holding the multiplicant at (S.1 0 1 1 1 1) = 47, and finding a number of the form (S.1 0 X X X) that produces a product just below 2048 will result in the determination of the shift limit. In this case 43 will produce a product of 2021 and 44 will produce a product of 2068. The limiting number that will require a shift will be S.1 0 1 1 0 2 = 43 and the difference in the multiplicand and the multiplier is 47 - 43 = 4. This indicates that the rest of the bits over the range of zero to four will determine if the shift is or is not necessary. The test can be made by addition and the test on the higher order bits X_1, X_2, and Y_1, Y_2 can be made by AND operations. Notice the form

```
  S.1 0 1 1 1 1
x S.1 0 1 1 1 1  ,
```

If X_2 and Y_2 are zero, then X_3 and Y_3 and X_4 and Y_4 must be 1's to create a no shift condition. Bits X_5,X_6 and Y_6 may change over a range of 4 and result in a no shift situation. Any other change in the last two bits in each number will result in a no shift condition.

```
  S.1 0 1 1 X X
x S.1 0 1 1 X X
```

12 — must add to less than 4 to give a no shift condition

AND, AND } must be 1's to reach a shift condition

Summary of Case 2 with X_2 and Y_2 equal to zero is

X_3	Y_3	X_4	Y_4	
1	1	1	1	} shift possible; any other combination results in no shift possible.

$X_5\ X_6$

$Y_5\ Y_6$

if sum > 4 → No shift;

if sum ≤ 4 → Shift.

In Case 3, the form is

$$\begin{array}{r} S.1\ 0\ X\ X\ \ldots \\ \times\ S.1\ 1\ Y\ Y\ \ldots \end{array} \quad \text{or} \quad \begin{array}{r} S.1\ 1\ X\ X \\ \times\ S.1\ 0\ Y\ Y \end{array} .$$

In this case, the shift limit is again 2^{2N-1}, and in the same 6-bit example

$$\begin{array}{rcr} S.1\ 0\ 0\ 0\ 0\ 0 & = & 32 \\ \times\ S.1\ 1\ 0\ 0\ 0\ 0 & = & \times\ 48 \end{array} .$$

By holding the 48 and adjusting the 32, the limit may be reached. For Case 2 it is found that since the add limit of the two numbers is 90, the multiply limit with 48 will be 42. To continue the same example with new form:

$$\begin{array}{rcr} S.1\ 0\ 1\ 0\ 1\ 0 & = & 42 \\ \times\ S.1\ 1\ 0\ 0\ 0\ 0 & = & \times\ 48 \end{array} .$$

The base required form is S.1 0 and S.1 1 which correspond to 32 and 48, which sum to 80. This implies that the last four bits of each number must add to 10 or less to require a shift and if they are lareger than 10, no shift is required.

The negative case may be studied in a similiar manner but the best approach to the overall problem is to only consider positive cases. The negative product obtained in a standard system is run through a two's complementer to get a positive product, then shifted to normal form and put back through the two's complementer to obtain a negative value in

normalized form. The tests may all be run with positive values; then, knowing the shift results, a negative product may be shifted without being two's complemented and tested. This result can be utilized with the shift results of the exponents to gate the output of the multiplication into the accumulating adder with only a small gate delay from the shift multiplexers.

Circuit Configuration with Look-Ahead Shift

The basic components of the floating point processor are the multiplier, exponent adder, accumulator adder, accumulator, and look-ahead shift system. These components are shown in Figure 18.

If the binary numbers X and Y are to be multiplied with this scheme, the mantissas and exponents are entered in the appropriate inputs. The look-ahead shift system determines if the result of the multiplication will require a shift for normalization and will complete its analysis long before the product is available. The two exponents must be added in a two's complement adder, and the result is compared with the exponent stored in the accumulator. The accumulator and new product exponents must match in magnitude before the addition and accumulation can take place. The results of the exponent addition will be available prior to the product of X and Y. Ample time is available to allow the exponent adjust system to adjust the shift multiplexer in coordination with the look-ahead shifter to preset the shift multiplexer and gate the product of X and Y into the adder in proper normalized form to allow addition and accumulation of the results.

At the onset of a new cycle, the accumulator is set to zero and the exponent adjust system is notified that a new cycle has begun. The first

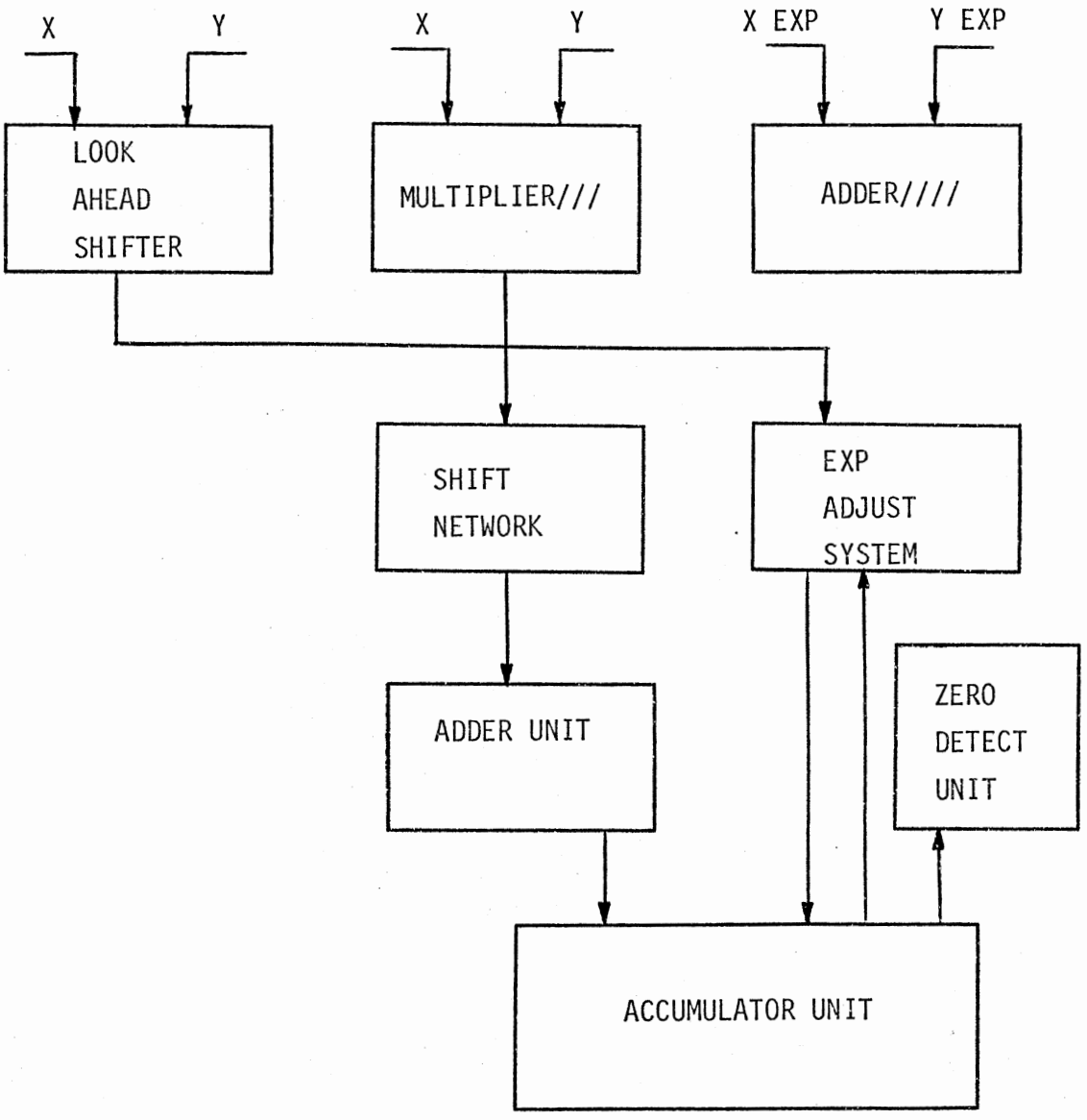


Figure 18. Floating Point Multiplier with Look-Ahead Unit

product will be added to the zero accumulator after the normalization shift requirement of the look-ahead shifter has been met. The remainder of the input data products to be accumulated will undergo normalization requirements generated by both the look-ahead shifter and exponent shifter. The conclusion of a row of the matrix can be determined by counting the number of inputs to the accumulators to determine if row calculations have been completed. This accumulation counting technique will be most acceptable to the system with the expansion to the optimal structure of processors.

Effects of Multiple Phase Operation on the Processing Element Design

The addition of faster or slower processing elements or individual multiplier units to the array structure will serve to complicate the problems of addition and accumulation of row terms in the process. This situation arises since the individual terms of a row of the matrix and the column of the vector, once multiplied, must all be summed to produce a term in the resulting vector. The required addition of these terms provides a problem area in the search for an efficient, inexpensive row processing scheme. One possibility is to use an adder and accumulator for each multiplier unit and therefore treat the design of all the processing elements as a standard structure. The hindrance in this scheme is that the accumulated data of the elements must further be added to produce the correct row values as shown in Figure 19.

This approach will require the first two accumulated values to be adjusted prior to addition to obtain the row value. Extra hardware is required to accomplish these tasks and extra time will elapse during

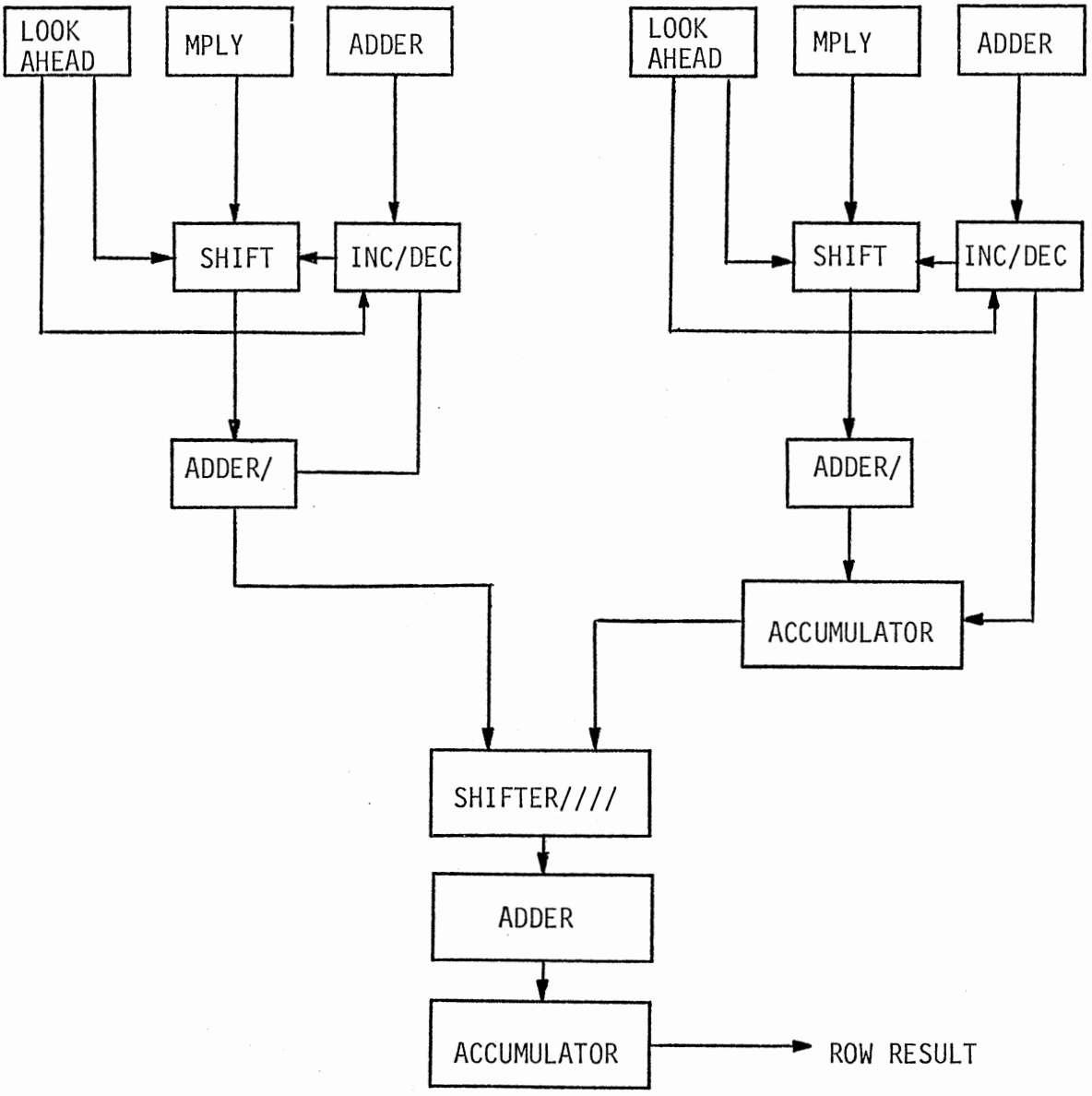


Figure 19. Processor Circuit with Two Processor Elements

these steps. An alternate approach is to combine the second multiplier into the first processing element while utilizing only one adder and accumulator to sum the row values in a time share mode, as shown in Figure 20.

An efficient time use of the adder and accumulator is possible due to the difference in operating speeds of the multipliers in the processing elements. Variations of both these schemes can be used as the requirements of the optimal solution results in different processing element requirements.

In the upper limiting case there exists one processor element for each term in the array, and a tree structure of adders may be required to compute the row values. As the requirements generated by the optimal solution change, resulting in slower processors being added to structure, it becomes conceivable to combine fast and slow processors in a given processing element. With further alterations in the optimal solution, it may not be possible to combine the results of a fast and slow multiplier concurrent with the next multiply time and the need for addition adders will result. The limiting factor in the process of combining more than one multiplier and shift unit to operate in conjunction with one adder and accumulator will be the number of additions that can be accomplished during the multiply period. For example, if an entire row of processing multipliers is used, one for each term in a row of the matrix, it may be possible to use one adder and accumulator to combine the results of all the elements using a time share data collection technique. This is acceptable provided that all the data terms can be accumulated in one multiply time.

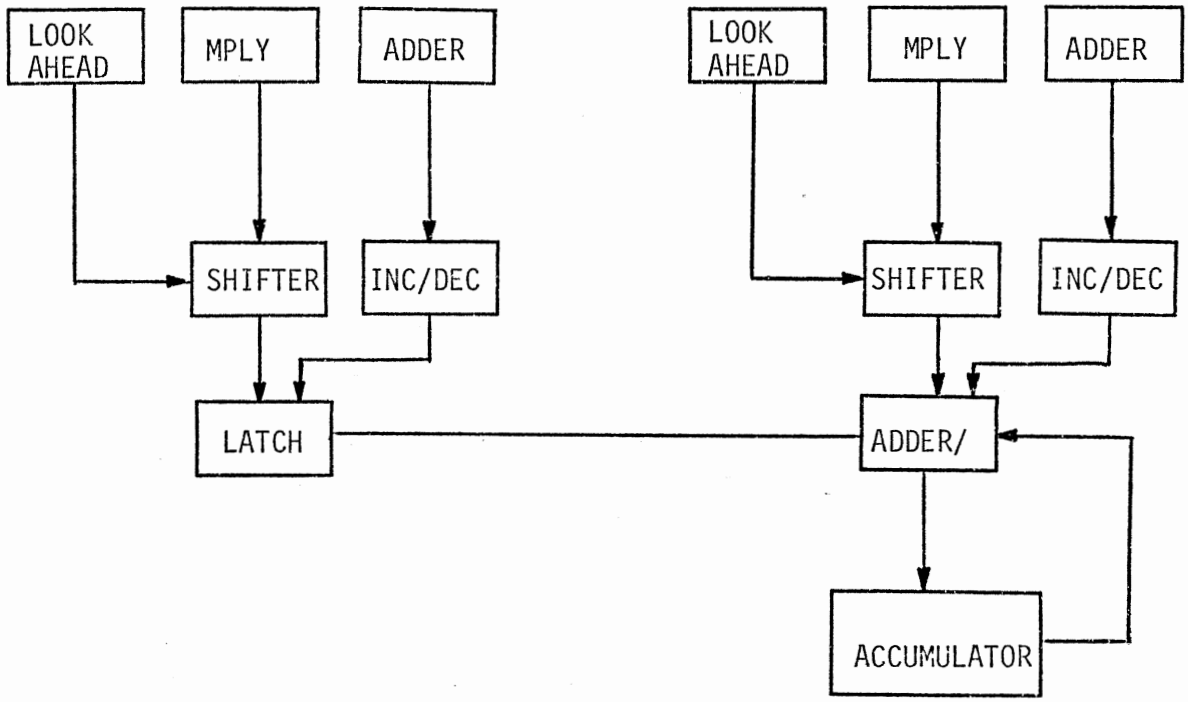


Figure 20. Floating Point Multiplier with Interleaving Design

Summary

The initial problem in the processor structure design is to determine the most feasible plan of calculating the terms of the matrix, vector problem, considering the problem structure. Certain techniques exist to facilitate the search for a practical solution to a minimal component processor configuration. Once the decision on a hardware structure internal to the processing elements is culminated, a further study into possible alterations caused by variations in the optimal solution of the processor configuration is of prime concern. The alterations generated by changing parameters in the problem requirements will result in possible complications or simplifications of the internal processor hardware.

In adherence with the design constraints of an array processor structure, certain improvements can be provided to a floating point hardware structure to improve its parallel operation. Knowledge of the binary number format and the problem algorithm aid the removal of the binary number normalization process from a serial configuration in hardware. The normalization processes resulting from multiplication and from addition can be preset concurrently with the multiplication of two terms at the start of each new cycle.

With the design constraints implemented in the structure and possible improvements to the circuitry generated, concern is shifted to the effects of existing hardware and the problems that standard parts might impose on the desired circuits. The effects of the multiple phase type structure and possible advantages and disadvantages of hardware were next considered and discussed. The final result of this chapter is that the design constraints and problem algorithm limitations have been carried through the system design process down to the lowest unit of the system.

CHAPTER VI

APPLICATION OF DISTRIBUTED ARCHITECTURE TO LINEAR RECURSIVE FILTERS

Introduction

One class of algotithm that employes a vector, matrix multiplication is the linear recursive filter, which is composed of both full and sub-optimal forms. This class of algorithm lends itself well to the properties of distributed architecture exhibited through the use of an optimal hareware design and its subsequent flexible throughput capability. This chapter will illustrate how a full order Kalman filter can be constructed in an optimal distributed architecture system.

Full Order Kalman Filter

A common problem in data transfer systems is the recovery of a pulse signal that has been corrupted by noise and has been distorted by being passed through a linear network such as a transmission line. Let a(t) be an input signal that has been corrupted by outside noise signals and subsequently passed throuth a system having a transfer function of b/(S + b). The output signal y(t) is further corrupted by measurement noise W(t), resulting in a signal r(t) which is the observed output signal of the modeled system. The transfer function of the transmission line is denoted to include all influences caused by linear distortion acting upon the input signal a(t).

The input signal a(t) will be a Manchester bi-phase pulse train of the type shown in Figure 21. This signal will be approximated with a Poisson-distributed-zero-crossing bi-level pulse train which has a well known autocorrelation function of the form $E_m^2 e^{-2k|\tau|}$ where E_m is the peak amplitude of the signal. The choice of the Poisson signal will allow an analytical approach to deturmine the transfer function of a linear system that will produce an approximation of the Manchester bi-phase signal at its output when its input is white noise. The transfer function of the signal generator and the transmission line are critical in the deturmination of the Kalman filter equations necessary to

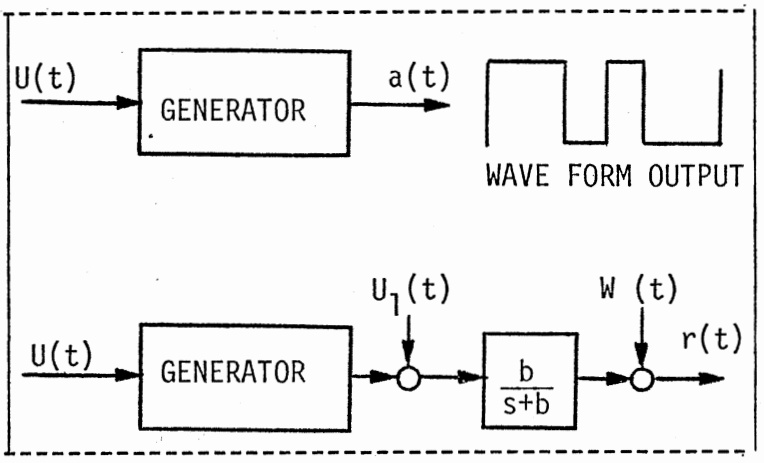


Figure 21. System Model Diagrams

process the Manchester bi-phase data. Figure 21 also illustrates the system model to include the transfer of the transmission line and the additive noise. The results of this system model will be a composit signal called r(t) that will provide the input to the Kalman Filter.

The power density spectrum of the random bi-level signal is given by

$$S_a(\omega) = E_m^2 \frac{2K}{(2K)^2 + \omega^2} \quad \text{where } \omega = 2\pi f.$$

The required linear system can be obtained by setting $S_a(\omega)$ equal to the product of its complex conjugates

$$S_a(\omega) = 2KE_m^2 \cdot \frac{1}{2K + j\omega} \cdot \frac{1}{2K - j\omega} .$$

$G^+(j\omega)$ is defined by the equation:

$$G^+(j\omega) = \frac{1}{2K + j\omega} .$$

The transfer function can be altered to arrive at a similar form

$$\frac{a}{S + a} \text{ , by multipling } S_a(\omega) \text{ by } \frac{4K^2}{4K^2} .$$

Be defining $G^+(j\omega)$ as

$$G^+(j\omega) = \frac{2K}{2K + j\omega} ,$$

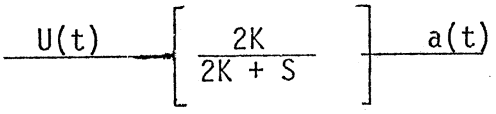
which results in

$$S_a(\omega) = |G_1^+(j\omega)|^2 S_{u1}(\omega),$$

the transfer function of the system can be determined. Utilizing this equation, the input power spectral density is computed by:

$$S_{u1}(\omega) = \frac{\frac{E_m^2 \, 2K}{[(2K)^2 + \omega^2]}}{\frac{(2K)^2}{(2K)^2 + \omega^2}} = 2\frac{E_m^2}{K} .$$

The random signal a(t) can be presented mathematically by the linear model



.

The total system model takes on the form shown in Figure 22 and b/(S + b) is the transfer function of the linear transmission media through which the signal is transmitted.

Kalman Filter Realization

Kalman filtering is based on the assumption that any random process can be modeled with a system which passes white noise through a linear circuit. The filter equations can be solved in the discrete form and values of the gain matrix can be generated for each sample time to be used by the system until it reaches steady state. In systems with a fast time period, the steady state values can be used to form the gain matrix and reduce the normal memory requirement of the computer.

The state variable form of the system will be:

$$\dot{X} = AX + B\bar{U}(t)$$

$$Z = CX + W(t).$$

U(t) and W(t) are assumed to be white noise processes such that:

$$E[U(t)U^{T}(\tau)] = Q\delta(t - r)$$

$$E[W(t)W^{T}(\tau)] = R\delta(t - r).$$

The state equations are determined to be:

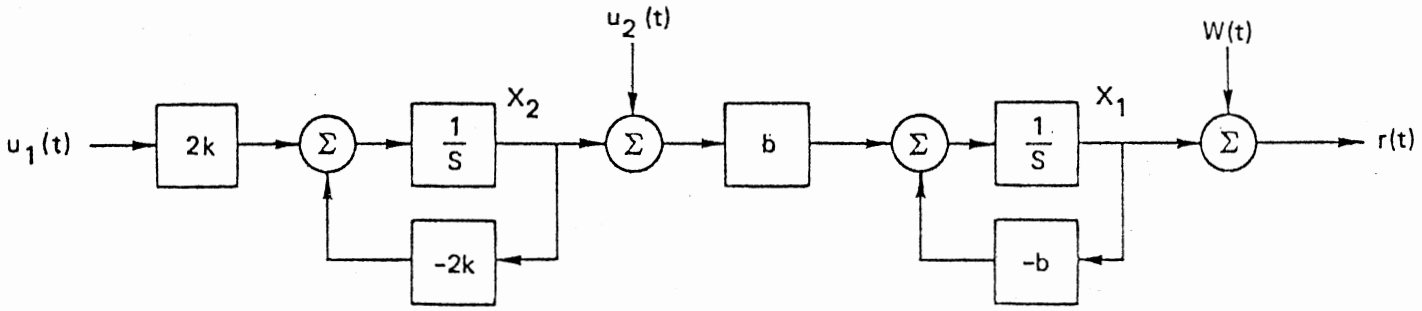


Figure 22. The Total Signal Generating Molel Diagram

$$\begin{bmatrix} \dot{\hat{X}}_1 \\ \dot{\hat{X}}_2 \end{bmatrix} = \begin{bmatrix} -b & b \\ 0 & -2K \end{bmatrix} \begin{bmatrix} \hat{X}_1 \\ \hat{X}_2 \end{bmatrix} + \begin{bmatrix} 0 \\ 2K \end{bmatrix} u_1(t)$$

$$Z = [1 \quad 0]\,\hat{X} + W(t)$$

$$Q = \frac{2E_m^2}{K} \qquad R = \frac{N_o}{2} .$$

The solution can be obtained from the equation

$$X(t) = \phi(t - t_o)X(t) + \int_{t_o}^{t} \phi(t - \tau)B(\tau)\bar{U}(\tau)d\tau .$$

For a discrete system

$$\text{let } t = t_{K+1}$$

$$t_o = t_K ,$$

Therefore,

$$X(t_{K+1}) = \phi(t_{K+1} - t_K)X(t_k) + \int_{t_K}^{t_{K+1}} \phi(t_{K+1} - \tau)B(\tau)\bar{U}(\tau)d\tau .$$

If a constant sampling rate of $T = t_{K+1} - t_K$ is assumed, then

$$X(t_{K+1}) = \phi(T)X(t_K) + W(t_K),$$

where

$$\bar{U}(t_K) \triangleq \int_{t_K}^{t_{K+1}} \phi(t_{K+1} - \tau)B(\tau)\bar{U}(\tau)d\tau .$$

The state transition matrix $\phi(t)$ is found by letting

$$\phi(t) = L^{-1}\left[(SI - A)^{-1}\right],$$

where

$$[SI - A]^{-1} = \begin{bmatrix} \frac{1}{S + b} & \frac{b}{(S + 2K)(S + b)} \\ 0 & \frac{1}{S + 2K} \end{bmatrix}.$$

The transform of the (1,2) term is found as follows:

$$L^{-1}\left[\frac{b}{(S + 2K)(S + b)}\right] = \frac{b}{(S + 2K)(S + b)} = \frac{A}{S + 2K} + \frac{B}{S + b}$$

$$A = \lim_{S \to -2K} \frac{b}{S + b} = \frac{b}{b - 2K}$$

$$B = \lim_{S \to -b} \frac{b}{S + 2K} = \frac{b}{2K - b}$$

$$L^{-1}\left[\frac{b}{(S + 2K)(S + b)}\right] = \frac{b}{b - 2K}\left[\frac{1}{S + 2K}\right] + \frac{b}{2K - b}\left[\frac{1}{S + b}\right]$$

$$= \frac{b}{b - 2K} e^{-2Kt} + \frac{b}{2K - b} e^{-bt}.$$

Therefore, the state transition matrix is:

$$\phi(t) = \begin{bmatrix} e^{-bt} & \frac{b}{b - 2K} e^{-2Kt} + \frac{b}{2K - b} e^{-bt} \\ 0 & e^{-2Kt} \end{bmatrix}$$

The mean value of the white noise driver is

$$E[U(K)] = \int_{t_K}^{t_{K+1}} \phi(t_{K+1} - \tau) B E[U(\tau)] d\tau = 0$$

and the covariance is found as follows:

$$E[U(K)U^T(K)] = E\left[\left[\int_{t_K}^{t_{K+1}} \phi(t_{K+1} - \tau)BU(\tau)d\tau\right] \cdot \left[\int_{t_K}^{t_{K+1}} \phi(t_{K+1}-\lambda)BU(\lambda)d\lambda\right]^T\right]$$

$$= E\left[\int_{t_K}^{t_{K+1}}\int_{t_K}^{t_{K+1}} \left[\phi(t_{K+1} - \tau)BU(\tau)U^T(\lambda)B^T\,{}^T(t_{K+1} - \lambda)\right]d\tau d\lambda\right]$$

$$= \int_{t_K}^{t_{K+1}}\int_{t_K}^{t_{K+1}} \left[\phi(t_{K+1}-\tau)BE\left(U(\tau)U^T(\tau)\right)B^T\phi(t_{K+1}-\lambda)\right]d\tau d\lambda$$

$$= \int_{t_K}^{t_{K+1}}\int_{t_K}^{t_{K+1}} \left[\phi(t_{K+1}-\tau)B_q\delta(\tau-\lambda)B^T\phi^T(t_{K+1}-\lambda)\right]d\tau d\lambda$$

$$= \int_{t_K}^{t_{K+1}} \phi(t_{K+1} - \tau)B_qB^T\phi^T(t_{K+1} - \tau)d\tau$$

$$= \int_{t_K}^{t_{K+1}} \phi(t_{K+1} - \tau)B_qB^T\phi^T(t_{K+1} - \tau)d\tau$$

The matrix computations leading to the evaluation of the Q matrix can be seen in Appendix D.

The discrete Kalman filter has a dynamic model of the form

$$X_{j+1} = \phi X_j + DU_j \quad \text{with } U_j \tilde{} N(0,Q_j)$$

and the obseration model is

$$Z_{j+1} = HX_{j+1} + W_j \quad \text{where } W_j \tilde{} N(0,R_j).$$

The algorithm for generating the discrete estimates for each sample time T is

$$X_{j+1} = X_{j+1|j} + K_{j+1}[Z_{j+1|T} - HX_{j+1|j}],$$

where

$$X_{j+1|j} = \phi X_j \quad \text{and} \quad X_o = \mu_o.$$

The gain values for each estimate are derived from the equation

$$K_{j+1} = P_{j+1|j} H^T [H P_{j+1|j} H^T + R_{j+1}]^{-1},$$

where

$$P_{j+1} = [I - K_{j+1} H] P_{j+1\ j}$$

$$P_o = Var[X_o] = \sigma_o^2 .$$

The algorithm is altered to a form more applicable to the distributed system by combining like terms to form:

$$X_{j+1} = \phi X_j + K_{j+1} Z_{j+1} - K_{j+1} H \phi X_j$$

$$X_{j+1} = \phi X_j - K_{j+1} H \phi X_j + K_{j+1} Z_{j+1}$$

$$X_{j+1} = [I - K_{j+1} H] \phi X_j + K_{j+1} Z_{j+1} \quad .$$

The values of the gain at each update are computed in off line simulation and stored in the system memory along with the terms of the Q matrix. In some problems of this type the steady state gain values are used at each update which allows less memory to be used to store data constants. The gain values of the Kalman filter problem are listed in Appendix C and it is evident that the gains reach a steady state value of KJ(1) = .215 and KJ(2) = 1.07. Further details of the program coding and format are shown in Appendix C.

Distributed Architecture Equation Format

The matrix equation for implementing the Kalman filter problem is of the form:

$$\begin{bmatrix} \hat{X}_{j+1(1)} \\ \hat{X}_{j+1(2)} \end{bmatrix} = \left(\begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} - \begin{bmatrix} K_1 \\ K_2 \end{bmatrix} \begin{bmatrix} H_1 & H_2 \end{bmatrix} \right) \begin{bmatrix} \phi_{11} & \phi_{12} \\ \phi_{21} & \phi_{22} \end{bmatrix} \begin{bmatrix} X_{j(1)} \\ X_{j(2)} \end{bmatrix} + \begin{bmatrix} K_1 \\ K_2 \end{bmatrix} Z_{j+1}$$

The value of the H vector is [1 0], indicating that this is a single observer filter. The terms of the equation can be further reduced to form the values of the constants to be stored in memory.

$$\begin{bmatrix} X_{j+1(1)} \\ X_{j+1(2)} \end{bmatrix} = \begin{bmatrix} (1 - K_1)\phi_{11} & (1 - K_1)\phi_{12} \\ -K_2\phi_{11} + \phi_{21} & -K_2\phi_{12} + \phi_{22} \end{bmatrix} \begin{bmatrix} X_{j(1)} \\ X_{j(2)} \end{bmatrix} + \begin{bmatrix} K_1 \\ K_2 \end{bmatrix} Z_{j+1}$$

and by combining the matrix equations into one matrix, vector multiplication, the proper form for implementation in hardware is reached.

$$\begin{bmatrix} X_{j+1(1)} \\ X_{j+1(2)} \end{bmatrix} = \begin{bmatrix} Y_{11} & Y_{12} & K_1 \\ Y_{21} & Y_{22} & K_2 \end{bmatrix} \begin{bmatrix} X_{j(1)} \\ X_{j(2)} \\ Z_{j+1|T} \end{bmatrix}$$

If the steady state gain values are used in a problem, the required memory for constants is reduced to the number of terms in the matrix. Such situations will further reduce the complexity of the design and directly affect the size, power, and cost of the system. In problems which to not lend themselves to this reduction process, a memory space is required to store a new set of values for each term in the matrix at each update calculation until steady state is reached.

Linear Program Formulation

The Kalman filter will be implemented with two types of processors, each with different operating characteristics.

	Cycle Time	Watts/Chip	Units Area/Chip
Processor 1	450 nsec	5 watts	3 units
Processor 2	800 nsec	1 watt	8 units

The time required for each update calculation is chosen to be one microsecond. At this point, the linear program equations can be formed and the boundaries set on power and size for the design by the following steps:

$$T_C = 1 \text{ microsecond}$$

$$T_1 = \text{largest integer } (T_C/T_{P1}) = \frac{1 \text{ } \mu\text{sec}}{500 \text{ nsec}} = 2$$

$$T_2 = \text{largest integer } (T_C/T_{P1}) = \frac{1\ \mu\text{sec}}{900\ \text{nsec}} = 1$$

$$P_1' = \text{matrix element}/T_1 = 6/2 = 3$$

$$P_2' = \text{matrix elements}/T_2 = 6/1 = 6$$

$$\text{Watts max} = P_1'(W_{P1}) = 3 \cdot 5 = 15$$

$$\text{Watts min} = P_2'(W_{P2}) = 6 \cdot 1 = 6$$

$$\text{Units max} = P_1'(U_{P1}) = 3 \cdot 3 = 9$$

$$\text{Units min} = P_2'(U_{P2}) = 6 \cdot 8 = 48.$$

The constant equations are now generated in the form:

$2P_1 + 1P_2 \leq 6$ time equation

$5P_1 + 1P_2 \geq X,\ 14 > X > 6$ power equation

$3P_1 + 8P_2 \geq X,\ 48 > X > 9$ area equation.

With a design of a second order filter the linear conditions and limits are easily plotted as shown in Figure 23. Figure 23 serves as a mapping of the design limits of the system under the constraints chosen up to this point and allows a more adequate choice of the power and unit area constraints to further simplify the design. To illustrate this simplification, let the power limit be equal to 14 watts and area limit equal 32 square units. Figure 27 shows the graph of this design, from which the optimal solution to the design can be obtained from the area above the time line and below both the power and unit area lines. The purpose of this step in the design is to check the feasibility of the constraints to determining if the design is possible prior to continuing the design sequence. Alterations in the power and area limits will increase or

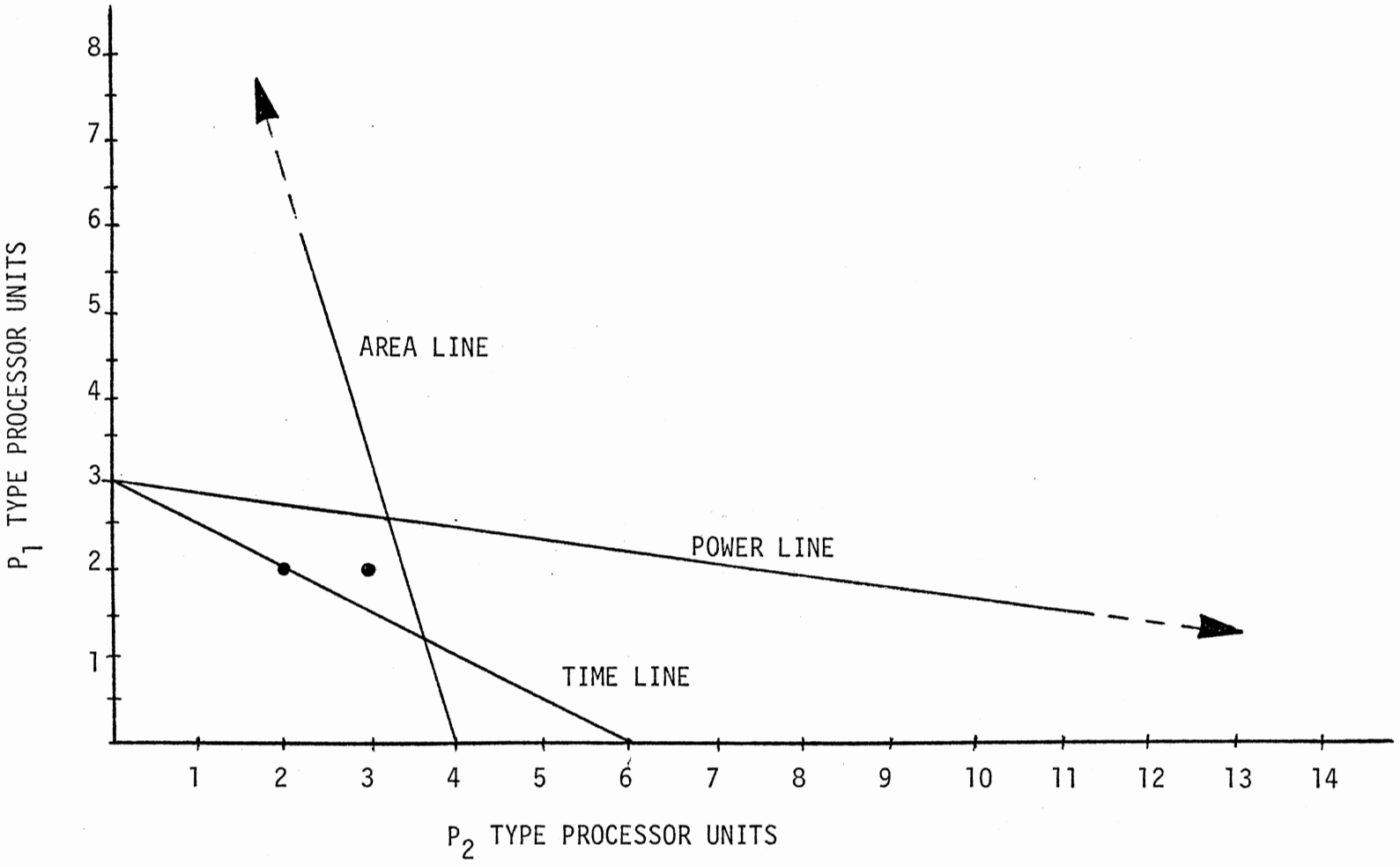


Figure 23. Plot of the Equations for the Kalman Filter Circuit

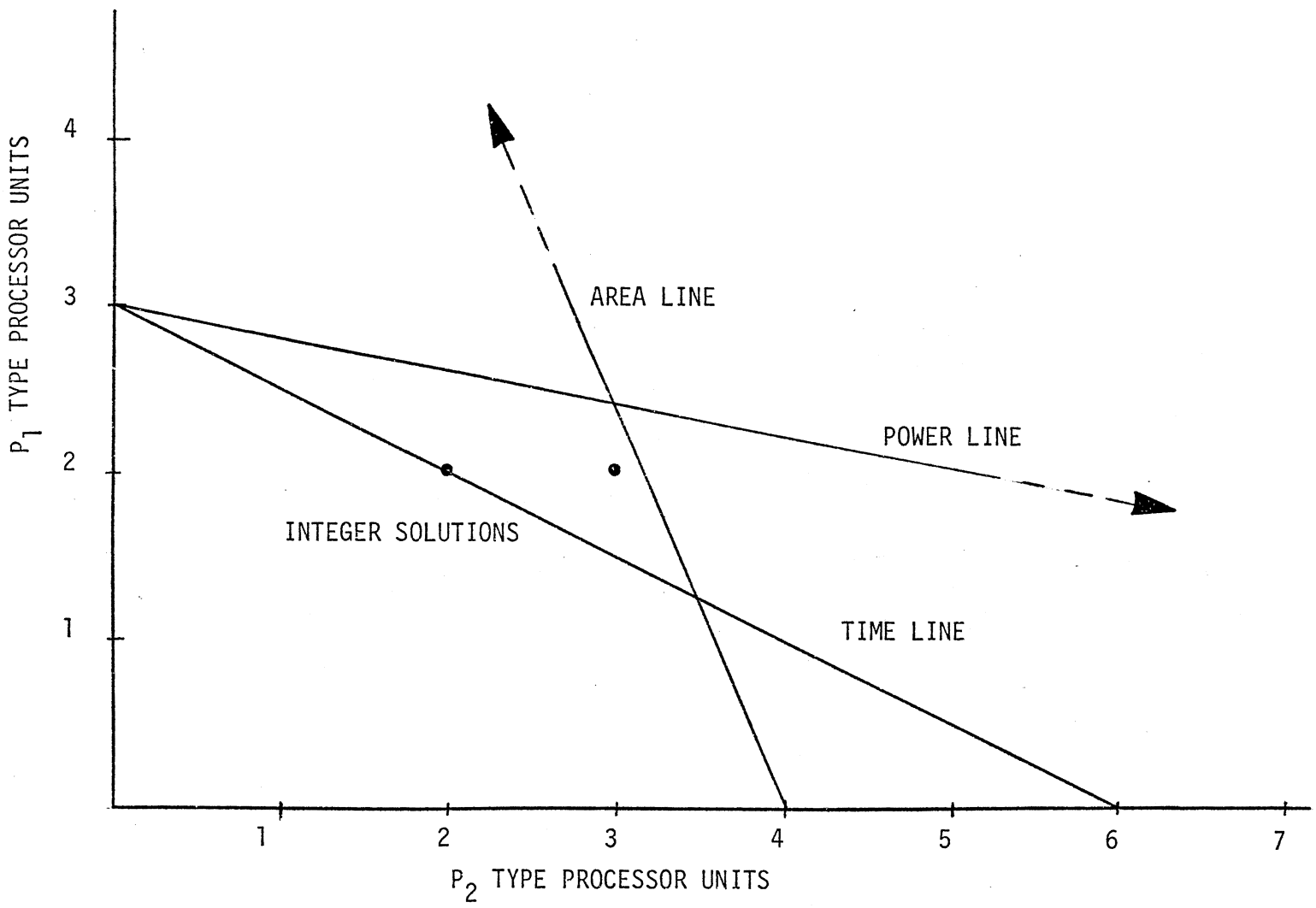


Figure 24. Integer Solutions to Kalman Filter Circuit

decrease the area from which the solution to the linear program can be located. The solution to the integer linear program is generated using a branch and bound technique and any reduction of the solution area will result in a reduction of the computation time to reach a solution.

The cost equation of the integer linear program is generated by letting C_1 equal the cost of a processor of type P_1 and C_2 is set equal to the cost of a type P_2 processor. Let $C_1 = 200$ and $C_2 = 50$. The resulting cost equation will then be

$$200\ P_1 + 50\ P_2 = Z.$$

The solution to the integer linear program may be obtained using the Land and Doig method as discussed in Appendix A or in this case a graphical solution is possible as seen in the plot in Figure 24. The results of the computer solution using the Land and Doig method are shown in Figure 25. The two solutions as seen in Figure 24 are $P_1 = 2$ and $P_2 = 2$ or $P_1 = 2$ and $P_2 = 3$. Using the cost equation of the program shows that the solution $P_1 = 2$ and $P_2 = 2$ will result in the minimum cost solution to the program with the constraint equations applied.

Circuit Design of Kalman Filter Problem

The results of the linear program solution are used in conjunction with Chapter IV to determine the configuration of the processor array and bus structure. In this problem, there will be two fast processors and two slow processors. Since the cycle time of the fast processor is 450 nsec, and the cycle time of the slow processor is 800 nsec, it is feasible to team one fast and one slow processor together in the interleaving design to compute one of the two rows of the matrix. Figure 26 shows a

```
 TEST PROBLEM

PRINT CONTROL PARAMETERS
  1   1

ROWS X COLUMNS AND NO. OF INTEGER VARIABLES

  4 X  3                                2

UPPER BOUND ON VARIABLE 1 TO N

 0.600D+01 0.600D+01

CONSTRAINT TYPES IN ROW ORDER
  1  -1  -1

MATRIX FORMAT CODE
  0

INPUT TABLEAU ECHO, CONSTRAINT VALUE LEFT.  BY ROW.

    0.0          0.200D+03    0.500D+02

    0.600D+01    0.200D+01    0.100D+01

    0.140D+02    0.500D+01    0.100D+01

    0.320D+02    0.300D+01    0.800D+01

INITIAL WORKING TABLEAU
     0            1            2

  0.0          0.20000D+03  0.500000D+02

  0.600000D+01 -0.200000D+01 -0.100000D+01

 -0.140000D+02  0.500000D+01  0.100000D+01

 -0.320000D+02  0.300000D+01  0.800000D+01

CONTINUOUS SOLUTION COMPLETE

FINAL TABLEAU FOR CONTINUOUS SOLUTION
     0           -3           -1

  0.42308D+03  0.76923D+01  0.11154D+03

 -0.35385D+01  0.15385D+00  0.23077D+00

 -0.43077D+01  0.23077D+00  0.28462D+01

 -0.12308D+01 -0.76923D-01 -0.61538D+00

OBJECTIVE FUNCTION =    423.0769231  AT ITERATION     2

STRUCTURAL VARIABLES: X(I)

I =   1         2

 0.123D+01 0.354D+01

OBJECTIVE FUNCTION =    500.0000000  AT ITERATION     8

STRUCTURAL VARIABLES: X(I)

I =   1         2
 0.200D+01 0.200D+01

OPTIMALITY ESTABLISHED

END OF PROBLEM, ITERATION NO.     9
```

Figure 25. Land and Doig Output Data

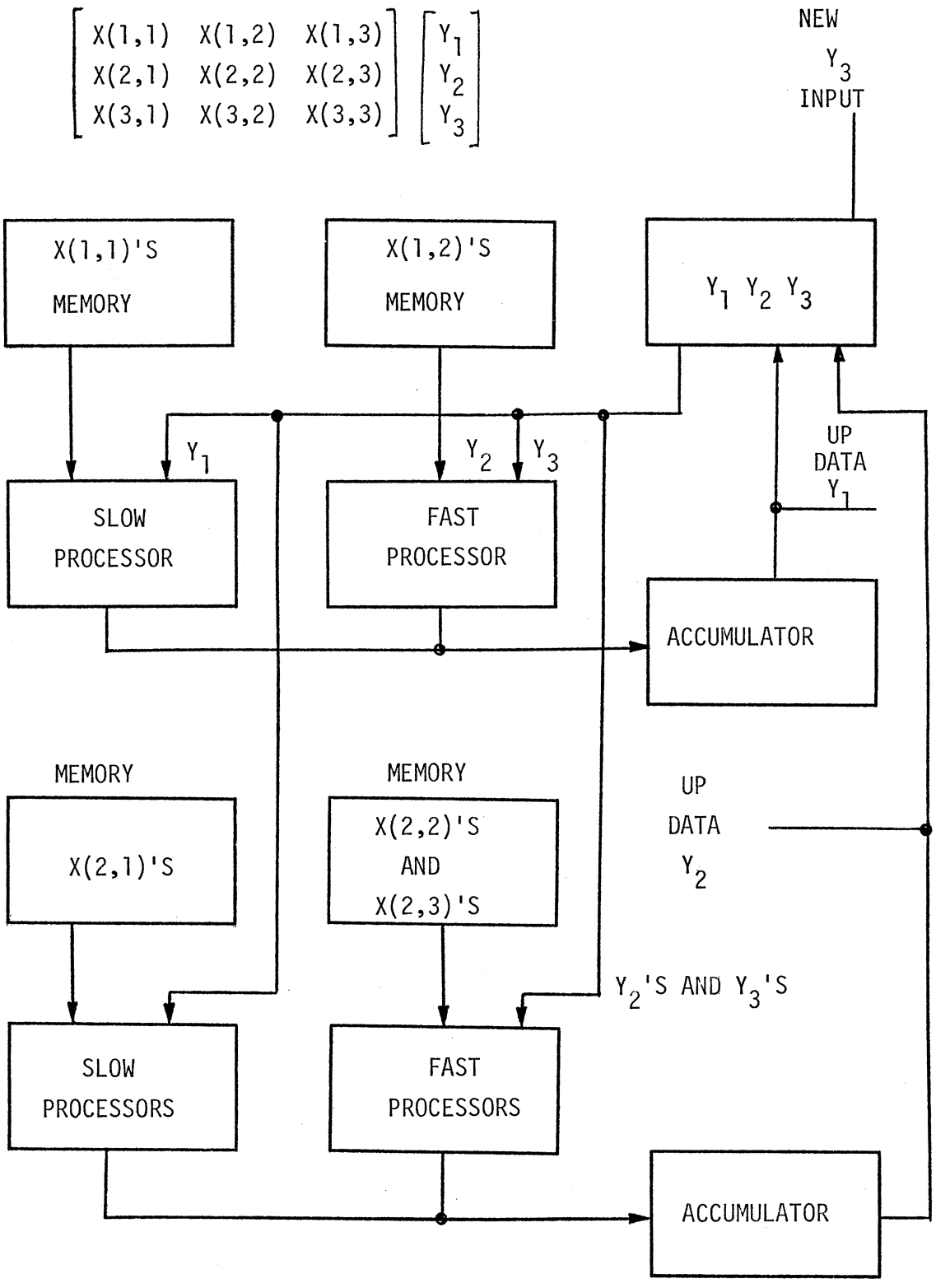


Figure 26. Data Flow for the Kalman Filter

possible circuit configuration based on this design. The slow processor and its memory will handle one term of the matrix and vector product during each cycle of the machine and the fast processor will handle two products during approximately the same time period with the results of both processors accumulated in one accumulation. Figure 27 illustrates in more detail the interleaving circuit operation to couple the fast and slow processor to compute a single row of the vector, matrix product.

Summary

This chapter has dealt with the formulation and design of a discrete Kalman filter using distributed architecture and circuit optimization. The system to be filtered was first modeled and the necessary calculations were performed to produce the Kalman filter equation and constants required to filter the system model. Next the distributed design was optimized to the design constraints and circuit block diagram were generated. This sequence of steps has served to illustrate the process of implementing an optimal architecture design for a common class of filtering problems.

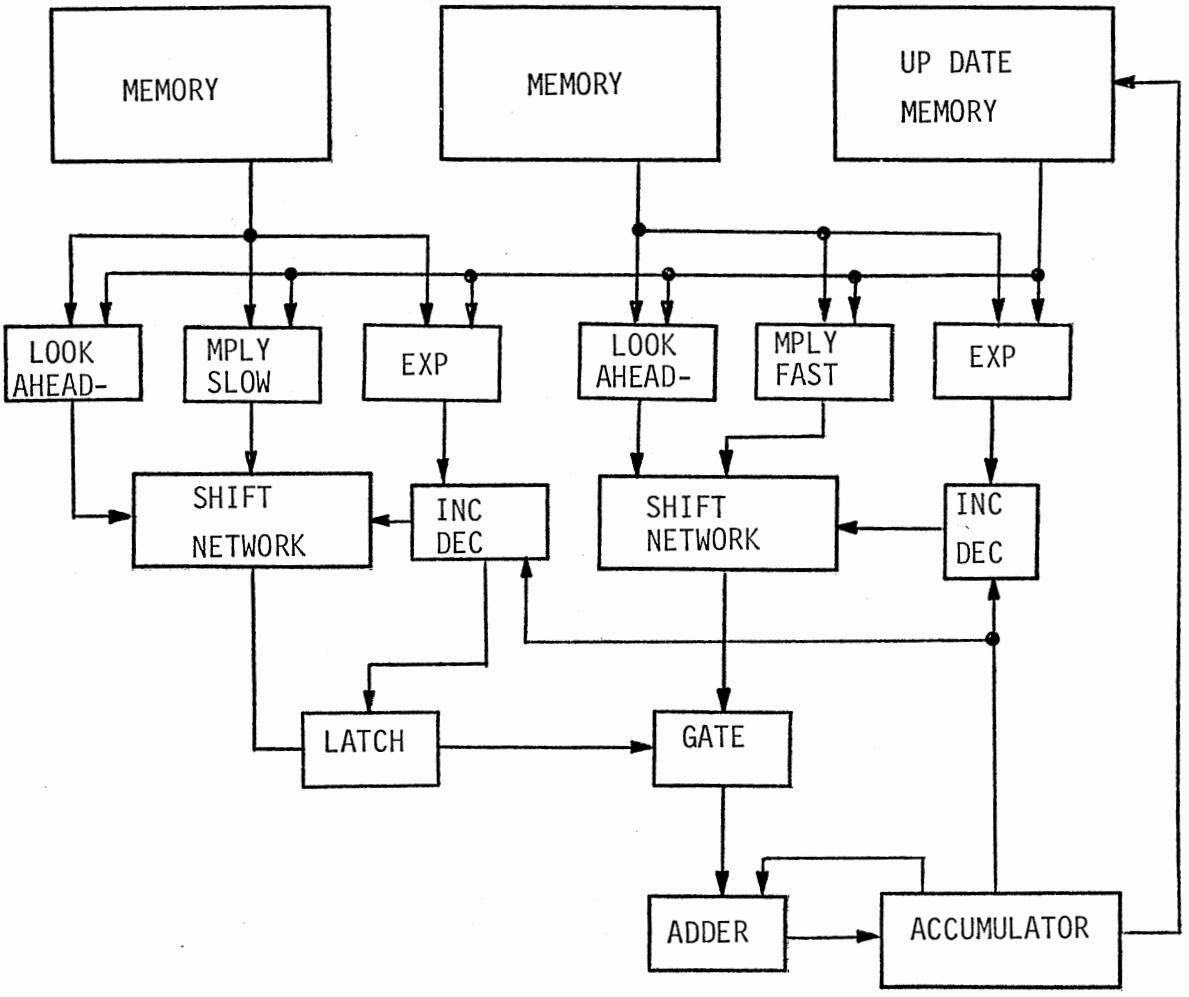


Figure 27. Circuit for the Kalman Filter

CHAPTER VII

CONCLUSIONS AND RECOMMENDATIONS

Conclusions

An algorithm for the design of special purpose distributed architecture computer optimized to cost and constrained by time, power, and circuit size has been described and implemented in this thesis.

A technique of using more than one processor to compute data during one time interval has become known as distributed processing and the structure of such systems is referred to as distributed architecture. With the reduction in size and cost of computing devices, special consideration is being given to design methods for computers that are based on the mathematical structure of a special class of problems. By designing the computer to take advantage of the structure of the problem classes of problems having common characteristics can be computed efficiently in a special-purpose machine. For a class of problems such as recursive linear filters, vehicle navigation, and sonar receiving, the common mathematical structure is the vector, matrix multiplication algorithm. The structure of vector, matrix problem has led to a distributed architecture known as array processing and specific criteria defining this class of data processing has been generating.

The desing of array processing machines in the past has been based on maximum throughput of data technically possible at the time of design. An

alternate concept has been presented in this thesis based on the optimization of the computing system with respect to cost and constrained by time, power, and circuit size. One of the fundamental questions in design is whether or not the system is capable of meeting the specifications of the design for a particular application. Optimization methods have been shown helpful in reducing the design of an array processor to obtain the best trade-off of speed, power, and circuit size with special emphasis placed on overall cost. From the concept of optimizing an array processor, has come a multi-phase processor design that utilizes different speed processors in a common circuit to better meet the overall design requirements. The basic criteria for the design has been generated from the definition of array processing and these rules of design can be followed through each phase of the design process as shown in Chapters IV and V.

Finally, the design algorithm has been put to use in Chapter VI to produce a multi-phased array processor to implement a Kalman filter. This same algorithm can be employed to generate a circuit structure for any similar structured problem that requires vector, matrix multiplications.

It is felt that the design of computers should, in certain classes of problems, be based on the problem strucutre or problem algorithm as well as the hardware present to construct the system. The entire system should be optimized with available optimization techniques to reduce the hardware structure to the best possible state.

Recommendations for Further Research

Within the design framework established by this thesis, several additional areas of research arise for the application of the proposed optimization design algorithm. The problems are for the most part concerned

with increasing the design to take into account all the parameters in the problem algorithm, the architecture structure, and the hardware components.

Various techniques for optimization and component reduction are present in the literature and with some modification these tools can be employed to aid in future system designs. The problems of cost, speed, power and size are being studied at the chip design level but little has been done to meet these problems at the system level. System design with the aid of computers is a fast growing area and large programs of an interactive nature are needed to speed the design of systems and take into account all the aspects of the problem at one time. There presently exists several software packages to do optimization problems as well as software to do some of the other steps in total system design and simulation. These software packages can be integrated together to form one design package capable of assisting a designer in producing a practical circuit to solve a problem.

The further changes in hardware can continue to affect the design considerations and processes. Floating point units are all but present today and their production will cause changes to the design of array processor systems as well as all other types of computing circuits. The interconnection of these future special-purpose chips will be of concern to design engineers in the future and better ways of using and keeping up with the new hardware must be found.

As the computer area and digital design area mature, more comples mathematical techniqes for the processing of data in real time applications will come into use. These methods will enhance the data processing capability of the future provided adequate means can be provided to design

and implement computers to perform these tasks. The primary area of concern for the design of these systems will still be speed, power, size, and cost of the system. Regardless of the new algorithms made available, the new chip produced, and the new problem to be solved, the constraints of cost, speed, power, and size are forever present.

SELECTED BIBLIOGRAPHY

1. Unger, S. H. "A Computer Oriented Towards Spatial Problems." IRE Proceedings, Vol. 36, October, 1958, pp. 1744-1750.

2. Holland, J. H. "A Universal Computer Capable of Executing an Arbitrary Number of Sub-Programs Simultaneously." Proceedings of the Eastern Joint Computer Conference, 1959, pp. 108-113.

3. Comfort, W. T. "A Modified Holland Machine." Proceedings, Fall Joint Computer Conference, 1963, pp. 481-488.

4. Slotnick, D. L. "The SOLOMON Computer." Proceedings, Fall Joint Computer Conference, 1962, pp. 97-107.

5. Gonzales, R. A. "A Multi-Layer Iterative Circuit Computer." IEEE Transactions on Electronic Computers, Vol. 12, December, 1963, pp. 781-790.

6. Barnes, G. H. "The ILLIAC IV Computer." IEEE Transactions on Computers, Vol. 17, August, 1968, pp. 746-757.

7. Pariser, J. J. and Maurer, H. E. "Implementation of the NASA Modular Computer with LSI Functional Characters." AFIPS Conference Proceedings, Fall Joint Computer Conference, 1969, pp. 231-245.

8. Dere, W. Y. and Sakrison, D. J. "The Berkeley Array Processor." IEEE Transactions on Computers, Vol. C-19, No. 5, May, 1970, pp. 444-447.

9. Cannon, L. E. "A Cellular Computer to Implement the Kalman Filter Algorithm." Ph.D. Thesis, Montana State University, August, 1969.

10. Weissberger, A. "Analysis of Multiple-Microprocessor System Architecture." IEEE Transactions on Computers, Vol. C-26, No. 5, June, 1977, pp. 151-163.

11. Bowra, J. W. and Torng, H. C. "The Modeling and Design of Multiple Function-Units Processors." IEEE Transactions on Computers, Vol. C-23, No. 4, March, 1976, pp. 210-221.

12. Flynn, M. J. "Some Computer Organizations and Their Effectiveness." IEEE Transactions on Computers, Vol. C-21, No. 9, September, 1972, pp. 948-960.

13. Enslow, P. H. Multiprocessors and Parallel Processing. New York: Wiley-Interscience, 1974.

14. Stone, A. L. "Parallel Computers." Introduction to Computer Architecture, SRA, 1975, pp. 318-374.

15. Lipovski, G. J. and Doty, K. L. "Developments and Directions in Computer Architecture." Computer, August, 1978, pp. 54-67.

16. Tang, C. K. "Cache System Design in the Tightly Coupled Multiprocessor System." Proceedings of the National Computer Conference, 1976, pp. 749-753.

17. Hayes, J. P. Computer Architecture and Organization. New York: McGraw-Hill Book Company, 1978.

18. Aho, A. V., Hopcroft, J. E. and Ullman, J. D. The Design and Analysis of Computer Algorithms. Mass.: Addison-Wesley, 1974.

19. Stratonovich, R. L. "Conditional Markov Processes." Theory of Probability and its Applications, Vol. 5, No. 2, 1960, pp. 156-178.

20. Storenson, H. W. and Stubberud, A. R. Linear Estimation Theory. National Technical Information Service, U. S. Department of Commerce, 1970, pp. 3-41.

21. Fenwick, P. M. "Binary Multiplication with Overlapped Addition Cycles." IEEE Transactions on Computers, Vol. C-21, No. 6, January, 1969, pp. 71-74.

22. McDonald, T. G. and Guha, R. K. "The Two's Complement Quasi-Serial Multiplier." IEEE Transactions on Computers, Vol. C-22, No. 4, December, 1975, pp. 1233-1235.

23. Waser, S. and Peterson, A. "Real-Time Processing Gains Ground with Fast Digital Multipliers." Electronics, September, 1977, pp. 93-99.

24. Wallace, C. S. "A Suggestion for a Fast Multiplier." IEEE Transactions on Electronic Computers, February, 1964, pp. 14-17.

25. MacSorley, O. L. "High Speed Arithmetic in Binary Computers." Proceedings of the IRE, January, 1961, pp. 67-91.

26. Parasuraman, B. "Hardware Multiplication Techniques for Microprocessor Systems." Computer Design, April, 1977, pp. 75-82.

27. Geist, D. J. "MOS Processors Pick-up Speed with Bipolar Multipliers." Electronics, July, 1977, pp. 113-115.

28. Pritchard, R. L. Trends in Integrated Electronics and Microprocessor Technology. General Electric Report No. 77CRD070, May, 1977.

29. Torng, H. C. and Wilhelm, N. C. "The Optimal Interconnection of Circuit Modules in Microprocessor and Digital System Design." IEEE Transactions on Computers, Vol. C-26, No. 5, May, 1977.

APPENDIXES

APPENDIX A

Mixed INTEGER LINEAR PROGRAM

Purpose

This programs finds the minimum of a multivariable, linear function subject to linear constraints, in which some or all of the variables may be restricted to integer values:

$$\text{Minimize } F=C_1X_1+C_2X_2+\ldots+C_{N1}X_{N1}+C_{N1}Y_{N1+1}+\ldots+C_NY_N$$

$$\text{Subject to } A_{ij}X_j+A_{ik}Y_k \;,=,\; B_i \qquad i=1,\ldots,m \quad j=1,\ldots,N1 \quad k=N1+1,\ldots,N$$

X_j are each integer and subject to an upper bound

$X_j,\ Y_k \quad 0.$

Method

The algorithm is based on the Land and Doig method. A dual simplex algorithm is imbedded in the program to obtain the starting, continuous solution and evaluate each integer trail. The specified integer variables are tested one at a time in paired values to establish direction and value. The algorithm is as follows:

1. The algorithm employs a dual simplex linear programming algorithm (not product form) hereinafter referred to as the LP. The tableau

is carried in compact Tucker form: the initial number of rows equals the number of problem constraints plus one; the initial number of columns equals the number of true variables plus one. Whenever a zero-constrained slack variable becomes non-basic, it is removed from the problem, resulting in a reduction by one of the number of columns in the tableau. Zero-constrained slack variables arise from two sources: equality constraints in the initial tableau; constraining a basic integer variable to an integer value (see 4 below). The number of rows in the tableau remains constant throughout.

2. Carry out an LP on the initial tableau. Print the solution. Check to see if all integer variables are integer valued. If so, the problem is terminated; if not, set the initial tolerance for the problem. (Tolerance is defined as the value below which the objective function must stay in order for a continuation of the current sequence of integer-constrained integer variables to be considered as a candidate for the mixed integer solution. Note that the objective function value at the continuous solution represents an absolute lower bound for the mixed integer solution.) Set to 1 the index of the integer variable being constrained.

3. Choose from those integer variables which are non-basic in the current tableau the one with highest coefficient in the objective function (shadow price). (The program makes use of the fact that the shadow price represents an underestimate of the increase in the objective function associated with constraining the non-basic integer variable to 1.) If no non-basic integer variable exists,

go to 4. Otherwise, store the current tableau and constrain the variable chosen to zero. This is done simply by removing the corresponding column from the tableau. (A non-basic variable is constrained to a non-zero integer value by adding the product to this value with each element in the corresponding column in the constant column of the tableau. The corresponding column is then removed from the tableau.) Go to 6.

4. Store the current tableau. Consider all integer variables X_i which are basic in the current tableau (there must be at least one) with value X_i^f. For each X_i determine the absolute difference between the increase in the objective function associated with the initial LP pivot step when X_i is constrained to $[X_i^f]$ and when X_i is constrained to $[X_i^f] + 1$. Choose as the integer variable to be constrained that X_i for which this difference is a maximum and constrain it to the value yielding the smaller increase. The actual constraining is accomplished by adding the integer value to the constant column of the row corresponding to to variable, and then stipulating that the row corresponds to a zero-constrained slack variable. Carry out an LP. If the objective function stays within the tolerance go to 6; otherwise go to 5.

5. If the current integer variable was constrained to $[X_i^f]$, record the fact that constraining it to values $[X_i^f] - k$ ($k = 1, 2, \ldots$) within its range need not be considered. Conversely, if X_i was set to $[X_i^f] + 1$, make note that values $[X_i^f] + 1 + k$ need not be considered. Go to 9.

6. Test the constrained variable index. If it is equal to N1, the number of integer variables in the problem, go to 9. Otherwise increase it by one and go to 3.
7. Decrease the constrained variable index by one and test it.
8. If it is zero go to 11. Otherwise go to 9.
9. Determine for the integer variable corresponding to the current value of the index whether its range has been exhausted (explicitly or implicitly) on neither, on one or on both sides of its current value. If it has been exhausted on both sides, go to 7. If the variable to be constrained has been exhausted on one side, constrain it to the unexhausted integer value closest to its current value in the proper direction. If the range is unexhausted on either side, determine in which direction to go using the method employed in 4, and proceed as for only that side open. (Note that the range of an integer variable which was non-basic when constrained is immediately exhausted from below.) Carry out an LP. If the objective function stays within the tolerance go to 6. Otherwise, note that the range of the current variable is exhausted in the direction in which its current value lies from its original value (see 5). Go to 9.
10. A better feasible mixed integer solution has beenobtained. Print the solution. Replace the tolerance by the objective function value. Go to 8.
11. For the current tolerance, all ranges of all the integer variables have been exhausted. If at least one feasible mixed integer solution has been obtained, the last printed solution is an optimal solution to the mixed integer problem and the problem is

terminated. Otherwise, the tolerance is increased, the continuous solution tableau is restored, the index of constrained integer variables is set to one, and control goes to 3.

If the program is terminated abnormally, the last printed feasible mixed integer solution (if any) is the best obtained. A flow diagram illustrating the above procedure is shown in Figure 28.

Program Description

1. Usage:

   The program consists of a main program only. Program size, solution estimate, and tableau coefficients along with control parameters are read in. The objective function to be minimized is the first row of the tableau.

2. Subroutines Required:

   None.

3. Description of Parameters:

   ISIZE Intermediate storage area = NZRTVR*(2*N-NZRTVR+1)/2 or as large as possible.

   NMRUNS Number of runs or problems to be solved.

   IOUT2 Print control for initial working tableau:

   0 = No print

   1 = Print tableau.

   IOUT3 Print control for continuous solution tableau:

   o = No print

   1 = Print tableau.

   IPACK Matrix format:

   0 = Unpacked, read all coefficients

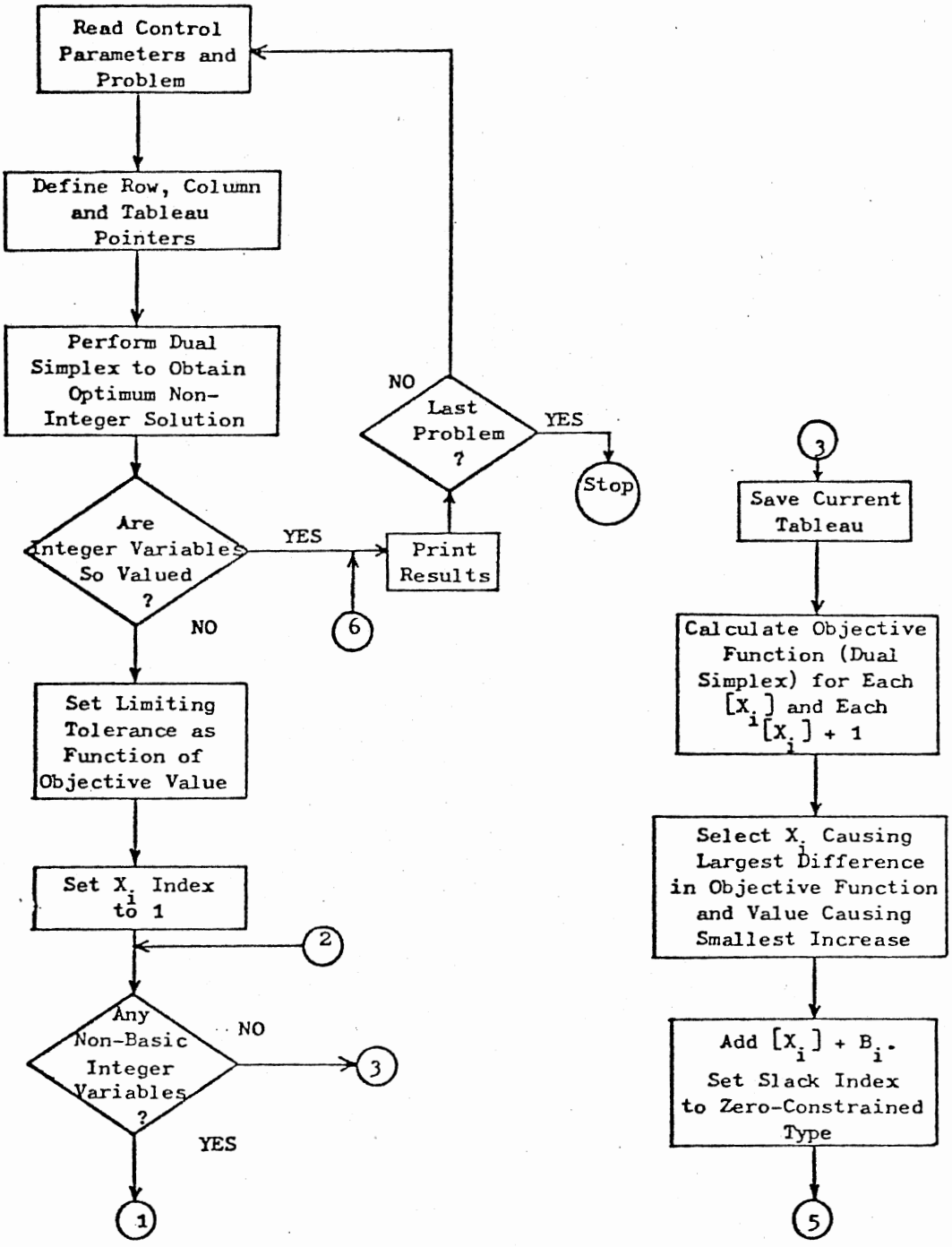


Figure 28. Mixed Integer Linear Programming Logic Diagram

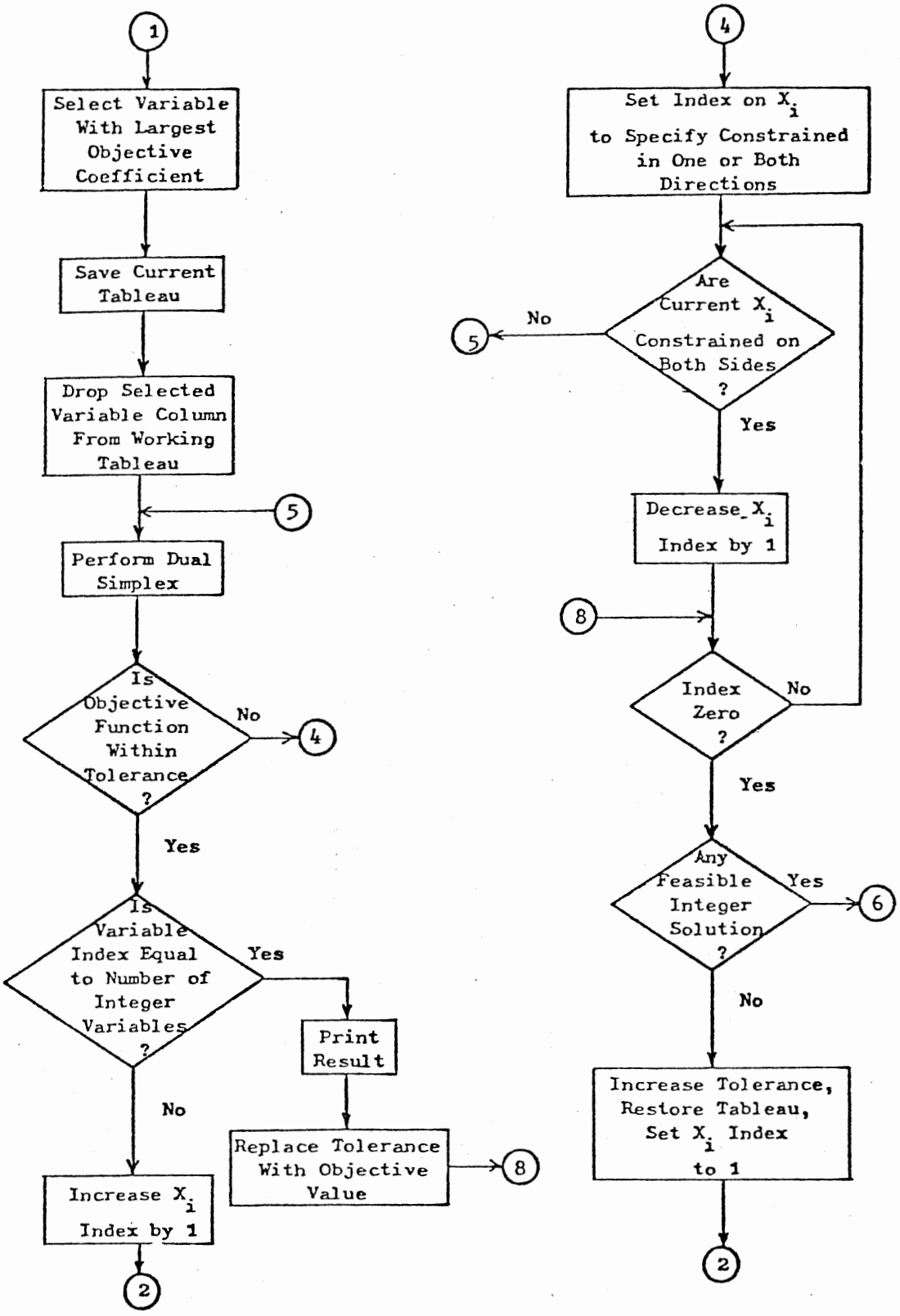


Figure 28. (Continued)

1 = Packed, read non-zero coefficients only.

SOLMIN Estimate of objective function if known, zero otherwise.

PCTTOL Tolerance as fraction of objective function for continuous solution (may be left at zero).

M Total number of rows.

N Total number of columns equals sum of X and Y variables plus 1 for constraints.

NM1 DO loop parameters: NM1 = N - 1.

NZR1VR Number of integer variables.

UPBND Vector of integer variable's upper bounds; size = N - 1.

IROW Vector of constraint types; size = M - 1:

+1 b_i

0 = b_i

-1 b_i.

ITEMP Column of coefficients being read in row i including objective row.

VAL Coefficient value of columns specified by ITEMP for row i.

ATAB Initial working tableau, N x M array.

NI Card reader unit number.

NO Printer unit number.

4. DIMENSION Requirements:

The COMMON* and DOUBLE PRECISION statements in the main program should be modified according to the requirements of the largest problem in the set being run. The parameters included in the following statements conform to the Input Parameter definitions

above: COMMON IROW(M), ITBROW(M), ICOL(N), ITBCOL(N), IVAR(N), ISVROW(M,NZR1VR), ISVRCL(NZR1VR), ICORR(NZR1VR), ISVN(NZR1VR), KSVN(NZR1VR+1), DOUBLE PRECISION ATAB(M+1,N), UPBND(N+1), TPVAL(NZR1VR+1), BTMVL(NZR1VR+1), VAL(NZR1VR+1), TBSAV(M,N), SAVTAB(M+1,NZR1VR*(2N-NZR1VR+1)/2), T(N).

5. Input Formats:

CARD TYPE	FORMAT	CONTENTS
1	(20I4)	ISIZE, NMRUNS
	(Appears only once per program execution.)	
2	(55H)	Problem title, identification
	(Put 1 in card column 1 for printer page control.)	
3	(20I4)	IOUT2, IOUT3, IPACK
4	(7E10.0)	SOLMIN, PCTTOL
5	(20I4)	M, N, NZR1VR
6	(7E10.0)	(UPBND(I), I=1, NM1)
	(If NZR1VR exceeds 7, additional CARD TYPE 6's required.)	
7	(20I4)	(IROW(I), I=2,M)
	(If M exceeds 20, additional TYPE 7's required.)	
8	If IPACK = 1	
	(7(I3, E7.0))	(ITEMP(K), VAL(K), K=1, 7)
	(If more than 7 non-zero coefficients exist, additional TYPE 8's required. Last TYPE 8 card must end with zero field. If last card full, insert blank card.)	
9	If IPACK = 0	
	(7E10.0)	(ATAB(I,J), J=1, N)

(one TYPE 9 per row including objective fct. If N exceeds 7, additional TYPE 9's per row required.)

6. Output:

The main program prints out the problem title supplied, print control parameters, problem size and number of integer variables, bounds on the integer bariables, codes for the constraint types, and the matrix format type code as part of the initial data.

The coefficient tableau is printed as raw data for checking purposes.

If IOUT2 = 1, the initial working tableau (as input to the first dual simplex solution) is printed in the Tucker form as used.

If IOUT3 = 1, the tableau from the continuous solution is printed.

The objective function value and values of each variable are printed for the continuous solution and for each feasible integer solution along with the present iteration number.

Error messages are printed for abnormal terminations suggesting the reason and giving the iteration number.

7. Summary of User Requirements:

a) Determine values for each problem set for SOLMIN, PCTTOL, M, N, NZR1VR, UPBND, IROW, NMRUNS, NI, and NO.

b) Calculate intermediate storage area for ISIZE.

c) Define code for matrix type for each problem.

d) Specify print control criteria for IOUT2, IOUT3.

e) Adjust COMMON size statements as needed to hold largest problem or satisfy machine limits.

f) Adjust FORMAT statements as necessary.

The FORTRAN program contained in this section is based on Branch and Bound Mixed Integer Programming, described on page 242 of "Catalog of Programs for IBM System 360 Models 25 and Above," GC 20-1619-8; program number 360D-15.2.005. Used by permission of International Business Machines Corporation.

APPENDIX B

LINEAR EQUATION BOUNDRY PLOT PROGRAM

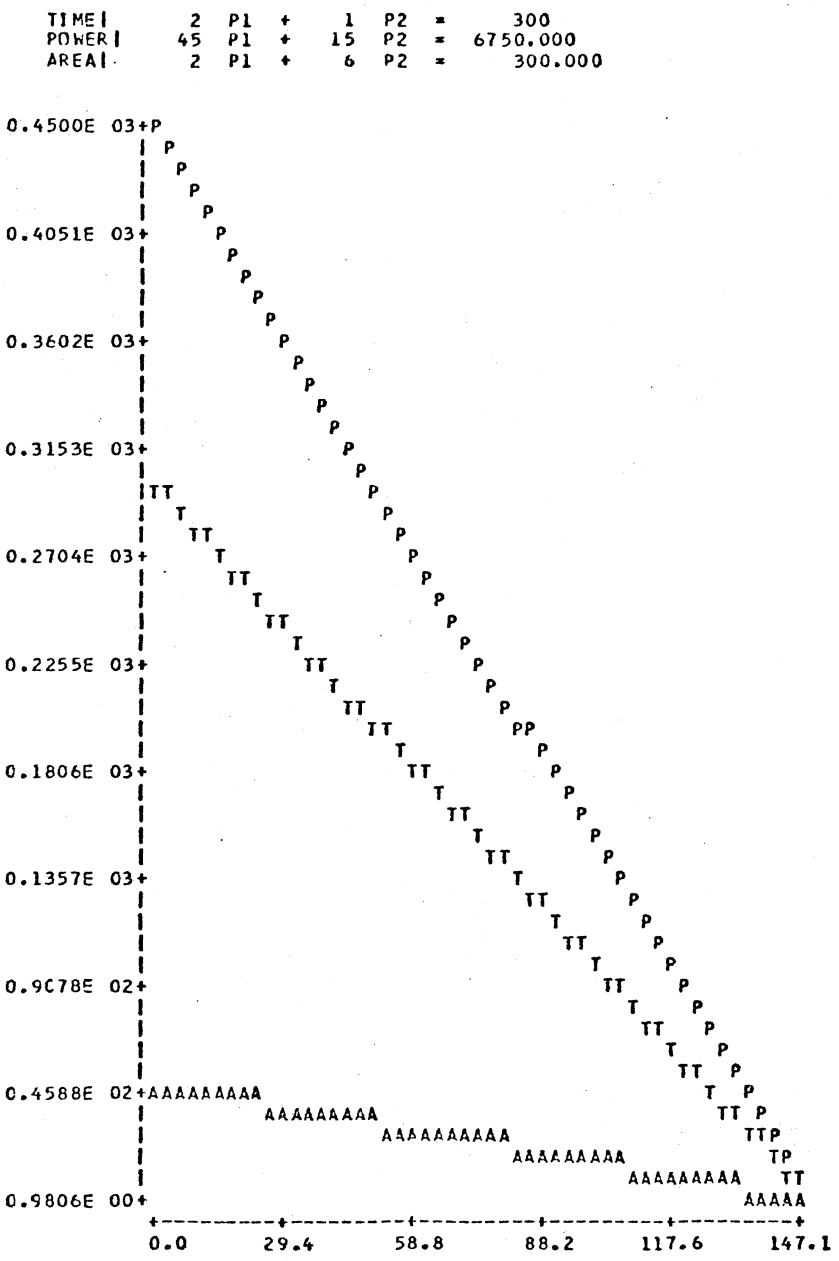
As discussed in Chapter 4, the following program was used to plot ans study the design parameters for a Multi-phase array processor circuit. The intent of the plotted data is to show the area in which the solution of the linear integer program is contained. This is done in such a way that alterations to the design can be introduced with ease. The program is written in Fortran and was executed on the IBM 370/158.

```
      $JOB TIME=10
 1          DIMENSION DATA1(153),DATA2(51),ISYMB(3)
 2          INTEGER M,N,SPEED1,SPEED2,POWER1,POWER2,UNITS1,UNITS2,TC
 3          INTEGER SIZE,DELAY,INTER,INTER2,LIMIT1,LIMIT2,POW1,POW2,UN1,UN2
      C*****************************************************
      C               DATA SECTION FOR PARAMETERS OF LINEAR EQUATIONS
      C
 4          DATA ISYMB/'T','P','A'/
 5          ZERO = 0.0
 6          M       = 15
 7          N       = 20
 8          SPEED1 = 500
 9          SPEED2 = 1000
10          POWER1 = 45
11          POWER2= 15
12          UNITS1 = 2
13          UNITS2= 6
14          TC      = 1000
15          INTER   =  TC / SPEED1
16          SIZE   = M  *  N
17          DELAY   = SPEED2 / SPEED1
18          INTER2   = INTER / DELAY
19          LIMIT1   = SIZE / INTER
20          LIMIT2 =   SIZE / INTER2
21          POW1   =  LIMIT1   * POWER1
22          POW2   =  LIMIT2   *  POWER2
23          UN1      =    LIMIT1   *  UNITS1
24          UN2      =    LIMIT2   *  UNITS2
25          ADDON   = FLOAT(IABS(POW1 - POW2)/4)
26          ADDUM   = FLOAT(IABS(UN1 - UN2)/4)
27          WATTS   =  FLOAT(POW1)
28          AREA     =  FLOAT(UN1)
      C
      C               CALCULATE THE DATA POINTS FOR THE EQUATIONS
      C
29          DO 60 K=1,4
30          POINT   =  FLOAT(LIMIT1)/51.0
31          POINT2    = 0.0
32          DO 20 J=1,51
33               DATA2(J) = POINT2
34               TP2 = (FLOAT(SIZE) - DATA2(J) * FLOAT(INTER))/FLOAT(INTER2)
35               DATA1(J) = TP2
36               WP2 = (WATTS - DATA2(J) * FLOAT(POWER1))/FLOAT(POWER2)
37               IF(WP2.LT.ZERO)WP2 = 0.0
38               DATA1(51+J) = WP2
39               UP2 = (AREA - DATA2(J) * FLOAT(UNITS1))/FLOAT(UNITS2)
40               IF(UP2.LT.ZERO)UP2 = 0.0
41               DATA1(102+J) = UP2
42               POINT2    = POINT2  + POINT
43          20 CONTINUE
44             WRITE(6,21)
45          21 FORMAT(1H1,3X,6HX-AXES,4X,6HY-TIME,4X,7HY-POWER,3X,6HY-AREA,/)
      C
      C               PRINT THE DATA POINTS OF X AND Y
      C
46             DO 30 I=1,51
47             WRITE(6,25)DATA2(I),DATA1(I),DATA1(I+51),DATA1(I+102)
48          30 CONTINUE
49          25 FORMAT(4F10.3)
50             WRITE(6,1)INTER,INTER2,SIZE
```

```
51          WRITE(6,2)POWER1,POWER2,WATTS
52          WRITE(6,3)UNITS1,UNITS2,AREA
53        1 FORMAT(1HL,10X,5HTIME|,4X,I3,2X,2HP1,2X,1H+,2X,I3,2X,2HP2,2X,1H=,2
           1X,I6)
54        2 FORMAT(1H ,10X,6HPOWER|,3X,I3,2X,2HP1,2X,1H+,2X,I3,2X,2HP2,2X,1H=,
           1F10.3)
55        3 FORMAT(1H ,10X,5HAREA|,4X,I3,2X,2HP1,2X,1H+,2X,I3,2X,2HP2,2X,1H=,2
           1X,F10.3,//)
56          CALL YPLOT(51,51,51,DATA2,DATA1,3,ISYMB,6)
57          WATTS = WATTS - ADDUM
58          AREA = AREA + ADDUM
59       60 CONTINUE
60          WRITE(6,500)
61      500 FORMAT(1H1)
62          STOP
63          END

64          SUBROUTINE YPLOT(IX,IY,NPNTS,X,Y,NCRVS,ISYMB,IOUT)
      C
      C     THIS IS A Y=F(X) PLOT ROUTINE THAT FEATURES:
      C        1. VARIABLE HEIGHT AND WIDTH OF PLOTS
      C        2. AUTOMATIC SCALING
      C        3. HORIZONTAL X AXIS AND VERTICAL Y AXIS
      C        4. MULTIPLE CURVES IN A SINGLE PLOT
      C
      C
      C     INPUT PARAMETERS
      C
      C     IX          NUMBER OF COLUMNS IN PLOT
      C     IY          NUMBER OF ROWS IN PLOT
      C     NPNTS       NUMBER OF DATA POINTS PER SET
      C     X           VECTOR OF X VALUES
      C     Y           VECTOR OF Y VALUES
      C     NCRVS       NUMBER OF CURVES TO BE PLOTTED
      C     ISYMB       VECTOR OF ALPHANUMERIC SYMBOLS TO BE USED IN PLOTS
      C     IOUT        LOGICAL UNIT NUMBER FOR OUTPUT DEVICE
      C
65          DIMENSION IGRPH(100,100),XSCAL(100),YSCAL(100),ISYMB(1),
           *          X(NPNTS),Y(1)
66          DATA IBLNK,IPLUS,MINUS/' ','+','-'/
      C
      C     INITIALIZE ARRAY TO BLANK
      C
67          DO 150 I=1,IX
68          DO 100 J=1,IY
69          IGRPH(I,J)=IBLNK
70      100 CONTINUE
71      150 CONTINUE
      C
      C     DETERMINE MINIMUM AND MAXIMUM X AND Y VALUES
      C
72          XMAX=X(1)
73          XMIN=X(1)
74          YMAX=Y(1)
75          YMIN=Y(1)
76          DO 200 I=2,NPNTS
77          IF(X(I).GT.XMAX) XMAX=X(I)
78          IF(X(I).LT.XMIN) XMIN=X(I)
79      200 CONTINUE
80          NYPTS=NCRVS*NPNTS
```

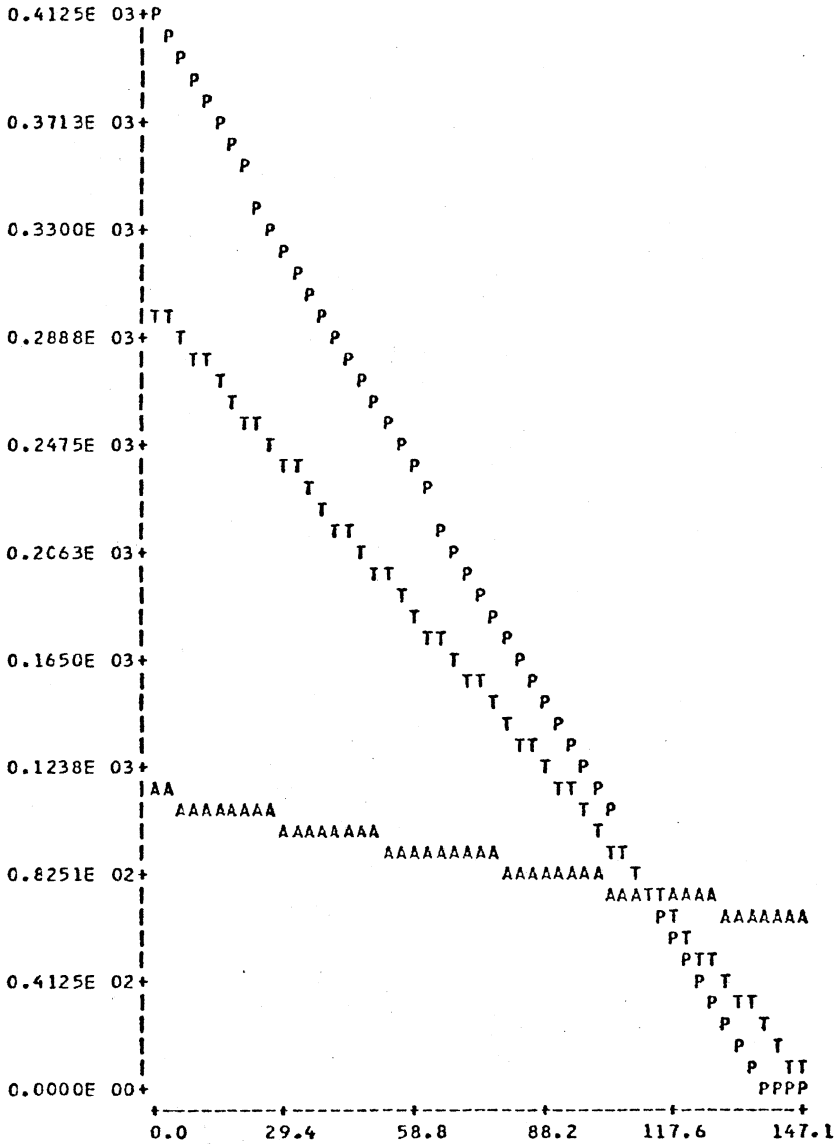
```
 81          DO 300 I=1,NYPTS
 82          IF(Y(I).GT.YMAX) YMAX=Y(I)
 83          IF(Y(I).LT.YMIN) YMIN=Y(I)
 84        300 CONTINUE
           C
           C     TEST FOR FLAT LINE
           C
 85          IF(YMAX.NE.YMIN) GO TO 250
 86          YHALF=YMAX/2.0
 87          IF(YHALF.EQ.0.0) YHALF=1.0
 88          YMAX=YMAX+YHALF
 89          YMIN=YMIN-YHALF
 90        250 CONTINUE
           C
           C     RECORD PLOT DATA IN ARRAY
           C
 91          DO 2000 J=1,NCRVS
 92          JCRV=NCRVS-J+1
 93          DO 1000 I=1,NPNTS
 94          ISUBX=IFIX((X(I)-XMIN)/(XMAX-XMIN)*FLOAT(IX-1)+.499)+1
 95          II=I+(JCRV-1)*NPNTS
 96          ISUBY=IFIX((Y(II)-YMIN)/(YMAX-YMIN)*FLOAT(IY-1)+.499)+1
 97          IGRPH(ISUBX,ISUBY)=ISYMB(JCRV)
 98        1000 CONTINUE
 99        2000 CONTINUE
           C
           C     COMPUTE SCALED VALUES FOR X AND Y
           C
100          DO 3000 I=1,IX
101          XSCAL(I)=FLOAT(I-1)/FLOAT(IX-1)*(XMAX-XMIN)+XMIN
102       3000 CONTINUE
103          DO 3050 I=1,IY
104          YSCAL(I)=FLOAT(I-1)/FLOAT(IY-1)*(YMAX-YMIN)+YMIN
105       3050 CONTINUE
           C
           C     PRINT OR DISPLAY ARRAY OF PLOTTED DATA
           C
106          DO 4000 I=1,IY
107          LINE=IY-I+1
108          K=LINE-1
109          IF(K/5*5.EQ.K)WRITE(IOUT,2)YSCAL(LINE),(IGRPH(J,LINE),J=1,IX)
110          IF(K/5*5.NE.K) WRITE(IOUT,1) (IGRPH(J,LINE),J=1,IX)
111       4000 CONTINUE
112          DO 5000 I=1,IX
113          IGRPH(I,1)=MINUS
114          IF((I-1)/10*10.EQ.I-1) IGRPH(I,1)=IPLUS
115       5000 CONTINUE
116          WRITE(IOUT,3) (IGRPH(J,1),J=1,IX)
117          WRITE(IOUT,4) (XSCAL(J),J=1,IX,10)
118          RETURN
119         1 FORMAT(18X,'|',100A1)
120         2 FORMAT(8X,E10.4,'+',100A1)
121         3 FORMAT(19X,100A1)
122         4 FORMAT(16X,10(F6.1,4X))
123          END

         C$IBSYS

         $ENTRY
```

X-AXES	Y-TIME	Y-POWER	Y-AREA
0.000	300.000	450.000	50.000
2.941	294.117	441.176	49.020
5.882	288.235	432.353	48.039
8.824	282.353	423.529	47.059
11.765	276.470	414.706	46.078
14.706	270.588	405.882	45.098
17.647	264.706	397.059	44.118
20.588	258.823	388.235	43.137
23.529	252.941	379.412	42.157
26.471	247.059	370.588	41.176
29.412	241.177	361.765	40.196
32.353	235.294	352.941	39.216
35.294	229.412	344.118	38.235
38.235	223.530	335.294	37.255
41.176	217.647	326.471	36.275
44.118	211.765	317.647	35.294
47.059	205.883	308.824	34.314
50.000	200.000	300.000	33.333
52.941	194.118	291.177	32.353
55.882	188.236	282.353	31.373
58.823	182.353	273.530	30.392
61.764	176.471	264.707	29.412
64.706	170.589	255.883	28.431
67.647	164.706	247.060	27.451
70.588	158.824	238.236	26.471
73.529	152.942	229.413	25.490
76.470	147.059	220.589	24.510
79.411	141.177	211.766	23.530
82.353	135.295	202.942	22.549
85.294	129.412	194.119	21.569
88.235	123.530	185.295	20.588
91.176	117.648	176.472	19.608
94.117	111.765	167.648	18.628
97.058	105.883	158.825	17.647
100.000	100.001	150.001	16.667
102.941	94.118	141.178	15.686
105.882	88.236	132.354	14.706
108.823	82.354	123.531	13.726
111.764	76.472	114.707	12.745
114.705	70.589	105.884	11.765
117.647	64.707	97.060	10.784
120.588	58.825	88.237	9.804
123.529	52.942	79.414	8.824
126.470	47.060	70.590	7.843
129.411	41.178	61.766	6.863
132.352	35.295	52.943	5.883
135.294	29.413	44.120	4.902
138.235	23.531	35.296	3.922
141.176	17.648	26.473	2.941
144.117	11.766	17.649	1.961
147.058	5.884	8.826	0.981


TIME| 2 P1 + 1 P2 = 300
POWER| 45 P1 + 15 P2 = 6750.000
AREA| 2 P1 + 6 P2 = 300.000
0.4500E 03
0.4051E 03
0.3602E 03
0.3153E 03
0.2704E 03
0.2255E 03
0.1806E 03
0.1357E 03
0.9078E 02
0.4588E 02
0.9806E 00
0.0
29.4
58.8
88.2
117.6
147.1

X-AXES	Y-TIME	Y-POWER	Y-AREA
0.000	300.000	412.533	112.500
2.941	294.117	403.709	111.520
5.882	288.235	394.886	110.539
8.824	282.353	386.063	109.559
11.765	276.470	377.239	108.578
14.706	270.588	368.416	107.598
17.647	264.706	359.592	106.618
20.588	258.823	350.768	105.637
23.529	252.941	341.945	104.657
26.471	247.059	333.122	103.676
29.412	241.177	324.298	102.696
32.353	235.294	315.475	101.716
35.294	229.412	306.651	100.735
38.235	223.530	297.827	99.755
41.176	217.647	289.004	98.775
44.118	211.765	280.181	97.794
47.059	205.883	271.357	96.814
50.000	200.000	262.534	95.833
52.941	194.118	253.710	94.853
55.882	188.236	244.887	93.873
58.823	182.353	236.063	92.892
61.764	176.471	227.240	91.912
64.706	170.589	218.416	90.931
67.647	164.706	209.593	89.951
70.588	158.824	200.769	88.971
73.529	152.942	191.946	87.990
76.470	147.059	183.122	87.010
79.411	141.177	174.299	86.029
82.353	135.295	165.475	85.049
85.294	129.412	156.652	84.069
88.235	123.530	147.829	83.088
91.176	117.648	139.005	82.108
94.117	111.765	130.182	81.128
97.058	105.883	121.358	80.147
100.000	100.001	112.535	79.167
102.941	94.118	103.711	78.186
105.882	88.236	94.888	77.206
108.823	82.354	86.064	76.226
111.764	76.472	77.241	75.245
114.705	70.589	68.417	74.265
117.647	64.707	59.594	73.284
120.588	58.825	50.770	72.304
123.529	52.942	41.947	71.324
126.470	47.060	33.123	70.343
129.411	41.178	24.300	69.363
132.352	35.295	15.476	68.383
135.294	29.413	6.653	67.402
138.235	23.531	0.000	66.422
141.176	17.648	0.000	65.441
144.117	11.766	0.000	64.461
147.058	5.884	0.000	63.481

```
TIME|      2  P1  +     1  P2  =      300
POWER|    45  P1  +    15  P2  =  6188.000
AREA|      2  P1  +     6  P2  =     675.000
```



APPENDIX C

KALMAN FILTER GAIN FINDING PROGRAM

This program simulates the Kalman Filter designed in Chapter 6 and is used to obtain the gain values at each up date point. These gain values will be ultilized in the memory of the Multi-phase array processor in on line operation.

```
      $JOB TIME=10
 1          DOUBLE PRECISION XJ1J(2),KH(2,2),R
 2          DOUBLE PRECISION EEE(50),WWW(50)
 3          DOUBLE PRECISION  XXX(2),FE(2,2),XJ(2),FEP(2,2),PJ(2,2)
 4          DOUBLE PRECISION  PJ1J(2,2),Q(2,2),HP(2),HPR,KJ(2),ZZ(25),Z,H(2)
 5          DOUBLE PRECISION XJP1(100),XJP2(100),KJP1(100),KJP2(100)
 6          DOUBLE PRECISION XJ1JP1(100),XJ1JP2(100)
 7          DOUBLE PRECISION K,T,B,QQ
 8          DOUBLE PRECISION A,C,D,E,F,G
 9          DOUBLE PRECISION YYY(50),RRR(50)
      C*****************************************************
      C               DATA SECTION
      C
10          DATA R/1.0D0/
11          DATA XJ/0.0D0,0.0D0/
12          DATA PJ/1.0D0,0.0D0,0.0D0,1.0D0/
      C*****************************************************
      C               INPUT DATA FROM MODEL
      C
13          ZZ(1) = 0.3D0
14          ZZ(2) = 0.45D0
15          ZZ(3) = 0.5D0
16          ZZ(4) = 0.55D0
17          ZZ(5) = 0.59D0
18          ZZ(6) = 0.6D0
19          ZZ(7) = 0.65D0
20          ZZ(8) = 0.7D0
21          ZZ(9) = 0.74D0
22          ZZ(10) = 0.739D0
23          ZZ(11) = 0.7388D0
24          ZZ(12) = 0.7388D0
25          ZZ(13) = 0.73666D0
26          ZZ(14) =0.7325D0
27          ZZ(15) = 0.7345D0
28          ZZ(16) = 0.748D0
29          ZZ(17) = 0.749D0
30          ZZ(18) = 0.75D0
31          ZZ(19) = 0.7511D0
32          ZZ(20) =0.746D0
33          ZZ(21) = 0.75D0
34          ZZ(22) = 0.7511D0
35          ZZ(22) = 0.7D0
36          ZZ(23) = 0.65D0
37          ZZ(24) = 0.55D0
38          ZZ(25) = 0.5D0
      C
      C*****************************************************
      C            FIND THE FEE AND Q MATRICES
39          B = 30.0D0
40          K = 27.0D0
41          T = 0.001D0
42          QQ = 1.0D0
43          A = ( B / ( B - 2.0D0 * K ))
44          C = (1.0D0 - DEXP( -4.0D0 * K * T ) )
45          D = ( B/(2.0D0 * K - B )  )
46          E = (1.0D0 - DEXP(-(2.0D0 * K + B) * T )  )
47          F = 2.0D0 * K + B
48          G = (2.0D0 * K**2 * D**2 * (1.0D0 - DEXP(-2.0D0 * B * T)) )/ B
49          Q(1,1)=( (K*A**2 * C) + (8.0D0 *K**2 * A * D * E)/F + G) * QQ
50          Q(1,2)= ( (K * A * C) + (4.0D0 * K**2 * D * E)/ F ) *QQ
```

```
51          Q(2,1) = Q(1,2)
52          Q(2,2) = (K * C ) * QQ
53          FE(1,1) = DEXP(-B * T )
54          FE(1,2) =(A * DEXP(-2.0D0 * K * T)) + (D * DEXP(-B * T) )
55          FE(2,1) = 0.0D0
56          FE(2,2) = DEXP(-2.0D0 * K * T)
      C*************************************************
      C
      C               WRITE THE FEE AND Q MATRICES
57          WRITE(6,50)Q(1,1),Q(1,2),Q(2,1),Q(2,2)
58          WRITE(6,50)FE(1,1),FE(1,2),FE(2,1),FE(2,2)
59       50 FORMAT(4D26.16)
60          I = 1
      C
      C               COMPUTE THE GAINS AND DATA POINTS
      C
61          DO 200 J=1,50
62               Z = ZZ(I)
63          IF(J.GT.25)Z = 0.0D0
      C
      C               FIND X(J+1|J)
      C
64          XJ1J(1) = FE(1,1) * XJ(1) + FE(1,2) * XJ(2)
65          XJ1J(2) = FE(2,1) * XJ(1) + FE(2,2) * XJ(2)
      C
      C               FIND P(J+1|J)
      C
66          FEP(1,1) = FE(1,1) * PJ(1,1) + FE(1,2) * PJ(2,1)
67          FEP(1,2) = FE(1,1) * PJ(1,2) + FE(1,2) * PJ(2,2)
68          FEP(2,1) = FE(2,1) * PJ(1,1) + FE(2,2) * PJ(2,1)
69          FEP(2,2) = FE(2,1) * PJ(1,2) + FE(2,2) * PJ(2,2)
70          PJ1J(1,1) = (FEP(1,1) * FE(1,1) + FEP(1,2) * FE(1,2) ) + Q(1,1)
71          PJ1J(1,2) = (FEP(1,1) * FE(2,1) + FEP(1,2) * FE(2,2) ) + Q(1,2)
72          PJ1J(2,1) = ( FEP(2,1) * FE(1,1) + FEP(2,2) * FE(1,2) ) + Q(2,1)
73          PJ1J(2,2) = ( FEP(2,1) * FE(2,1) + FEP(2,2) * FE(2,2) ) + Q(2,2)
      C
      C               FIND GAIN VALUES
      C
74          KJ(1) = PJ1J(1,1) / (PJ1J(1,1) + R)
75          KJ(2) = PJ1J(2,1) / (PJ1J(1,1) + R)
      C
      C               KALMAN FILTER EQUATIONS
      C
76          XJ(1) = XJ1J(1) + KJ(1) * ( Z - XJ1J(1))
77          XJ(2) = XJ1J(2) + KJ(2) * ( Z - XJ1J(1))
      C
      C               FIND P(J+1) FOR NEXT UP DATE
      C
78          PJ(1,1) = (1.0D0 - KJ(1)) * PJ1J(1,1)
79          PJ(1,2) = (1.0D0 - KJ(1)) * PJ1J(1,2)
80          PJ(2,1) = -KJ(2) * PJ1J(1,1) + PJ1J(2,1)
81          PJ(2,2) = -KJ(2) * PJ1J(1,2) + PJ1J(2,2)
      C*************************************************
      C               PLACE THE DATA IN ARRAYS FOR PRINTING
      C
82          XJP1(J) = XJ(1)
83          XJP2(J) = XJ(2)
84          KJP1(J) = KJ(1)
85          KJP2(J) = KJ(2)
86          XJ1JP1(J) = XJ1J(1)
```

```
87          XJ1JP2(J) = XJ1J(2)
88          YYY(J) = PJ(1,1)
89          RRR(J) = PJ(1,2)
90          WWW(J) = PJ(2,2)
91          EEE(J) = PJ(2,1)
92          I = I + 1
93          IF(I.GT.25) I = 1
94        200 CONTINUE
      C*************************************************
      C               PRINT SECTION
      C
95          WRITE(6,500)
96        500 FORMAT(1H1,10X,'XJ(1)',22X,'XJ(2)',22X,'KJ(1)',22X,'KJ(2)',//)
97          DO 300 J=1,50
98                WRITE(6,100)XJP1(J),XJP2(J),KJP1(J),KJP2(J)
99        100 FORMAT(4D26.16)
100       300 CONTINUE
101         WRITE(6,600)
102       600 FORMAT(1H1,10X,'XJ1J(1)',20X,'XJ1J(2)',//)
103         DO 400 J=1,50
104               WRITE(6,110)XJ1JP1(J),XJ1JP2(J)
105       110 FORMAT(2D26.16)
106       400 CONTINUE
107         WRITE(6,803)
108         DO 700 J=1,50
109         WRITE(6,801)YYY(J),RRR(J),EEE(J),WWW(J)
110       801 FORMAT(4D26.16)
111       700 CONTINUE
112       803 FORMAT(1H1)
113         STOP
114         END

      C$IBSYS

      $ENTRY
```

KJ(1)	KJ(2)
0.48544148845483140 00	0.35141232514324130-01
0.31646519953726640 00	0.11830723604873170 00
0.23690407470461910 00	0.23969771249048480 00
0.19615139612466800 00	0.38760287923121600 00
0.17718526847508310 00	0.54784017550617940 00
0.17189515946394690 00	0.70476248318071230 00
0.17505012956368770 00	0.84389230697120460 00
0.18265203376158440 00	0.95507182639296370 00
0.19166240567097340 00	0.10343889518874460 01
0.20005920670443990 00	0.10838636531276760 01
0.20679421325753130 00	0.11093631091378700 01
0.21157699654696260 00	0.11180757148310860 01
0.21458401412901250 00	0.11166180152565790 01
0.21620182352405880 00	0.11101045607930600 01
0.21685404480059400 00	0.11019790472339620 01
0.21690989085533670 00	0.10942575542405080 01
0.21665035896000150 00	0.10879093106550700 01
0.21626738239732970 00	0.10832228556211600 01
0.21587815032797500 00	0.10800971400173130 01
0.21554399931680890 00	0.10782473111427990 01
0.21528843145524190 00	0.10773363759080510 01
0.21511193609356530 00	0.10770501794739000 01
0.21500300582725690 00	0.10771327097227380 01
0.21494560126727290 00	0.10773959216763190 01
0.21492369775800240 00	0.10777149235427220 01
0.21492365309027540 00	0.10780162131213850 01
0.21493509164122830 00	0.10782639808142480 01
0.21495088286287350 00	0.10784473979721430 01
0.21496664920317980 00	0.10785702863888820 01
0.21498010079227260 00	0.10786435560698360 01
0.21499037826955320 00	0.10786802077435940 01
0.21499749742767560 00	0.10786924147494450 01
0.21500192947560360 00	0.10786901225408550 01
0.21500431438508730 00	0.10786806473993580 01
0.21500528620214130 00	0.10786688556841610 01
0.21500538267066280 00	0.10786576192372980 01
0.21500501223291700 00	0.10786483473733250 01
0.21500445585389040 00	0.10786414801136870 01
0.21500388675028640 00	0.10786368883041710 01
0.21500339655056480 00	0.10786341660687040 01
0.21500302095070010 00	0.10786328239452740 01
0.21500276132276060 00	0.10786324017397650 01
0.21500260102623170 00	0.10786325228866020 01
0.21500251655235690 00	0.10786329103164810 01
0.21500248432272260 00	0.10786333799415690 01
0.21500248428220210 00	0.10786338234591750 01
0.21500250112681780 00	0.10786341881470400 01
0.21500252437129760 00	0.10786344580901510 01
0.21500254757506910 00	0.10786346389380850 01
0.21500256737039120 00	0.10786347467621200 01

APPENDIX D

MATRIX COMPUTATIONS OF Q MATRIX

The evaluation of the terms of the Q matrix started in Chapter 6 are continued in the following appendix. The solution begins with the matrix operations shown in figure 29 and is followed by a term by term integration of the matrix parts leading to the solution of the Q matrix.

$$= q \int_0^T \left[\begin{bmatrix} e^{-b(t-\tau)} & Ae^{-2K(t-\tau)} + Be^{-b(t-\tau)} \\ 0 & e^{-2K(t-\tau)} \end{bmatrix} \begin{bmatrix} 0 \\ 2K \end{bmatrix} \begin{bmatrix} 0 & 2K \end{bmatrix} \begin{bmatrix} e^{-b(t-\tau)} & 0 \\ Ae^{-2K(t-\tau)} + Be^{-b(t-\tau)} & e^{-2K(t-\tau)} \end{bmatrix} \right] d\tau$$

$$= q \int_0^T \left[\begin{bmatrix} 0 & 4K^2 Ae^{-2K(t-\tau)} + Be^{-b(t-\tau)} \\ 0 & 4K^2 e^{-2K(t-\tau)} \end{bmatrix} \begin{bmatrix} e^{-b(t-\tau)} & 0 \\ Ae^{-2K(t-\tau)} + Be^{-b(t-\tau)} & e^{-2K(t-\tau)} \end{bmatrix} \right] d\tau$$

$$= q \int_0^T \begin{bmatrix} 4K^2 Ae^{-2K(t-\tau)} + Be^{-b(t-\tau)}{}^2 & 4K^2 e^{-2K(t-\tau)} Ae^{-2K(t-\tau)} + Be^{-b(t-\tau)} \\ 4K^2 e^{-2(t-\tau)} Ae^{-2K(t-\tau)} + Be^{-b(t-\tau)} & 4K^2 e^{-4K(t-\tau)} \end{bmatrix} d\tau \ .$$

Figure 29. Matrix Computations for the Q Matrix

Evaluating the first term of the Q matrix gives

$$q_{11} = q \int_0^T 4K^2\left(Ae^{-2K(t-\tau)} + Be^{-b(t-\tau)}\right)^2 d\tau$$

where

$$A = \frac{b}{b - 2K} \quad \text{and} \quad B = \frac{b}{2k - b} .$$

Part 1:

$$= a\ 4K^2A^2 \int_0^T e^{-4KT + 4K\tau}\, d\tau = 4K^2A^2e^{-4KT}\ \frac{e^{4KT}}{4K}\Bigg|_0^T$$

$$= KA^2\left(1 - e^{-4KT}\right)$$

Part 2:

$$= q\ 8K^2ABe^{-(2K + b)T} \int_0^T e^{(2K + b)\tau}\, d\tau$$

$$= \frac{8K^2AB}{2K + b}\left(1 - e^{-(2K + b)T}\right)$$

Part 3:

$$= 4K^2B^2e^{-2bT} \int_0^T e^{2b\tau}\, d\tau = \frac{4K^2B^2}{2B}\ 1 - e^{-2bT}$$

$$= q\ \frac{2K^2B^2}{b}\left(1 - e^{-2bT}\right).$$

Terms $q_{12} = q_{21}$ and is evaluated by the equation

$$q_{12} = q \int_0^T 4K^2 e^{-2K(t-\tau)}\left(Ae^{-2K(t-\tau)} + Be^{-b(t-\tau)}\right) d\tau \ .$$

Part 1:

$$q\ 4K^2Ae^{-4KT} \int_0^T e^{4K\tau} d\tau = \frac{4K^2A}{4K}\left(1 - e^{-4KT}\right) = KA\left(1 - e^{-4KT}\right)$$

Part 2:

$$q\ 4K^2Be^{-(2K+b)T} \int_0^T e^{(2K+b)\tau} d\tau = q\ \frac{4K^2B}{2K+b}\left(1 - e^{-(2K+b)T}\right).$$

Term q_{22} is determined to be

$$q_{22} = q\ 4K^2e^{-4KT} \int_0^T e^{4K\tau} d\tau = \frac{4K^2}{4K}\left(1 - e^{-4KT}\right) = K\left(1 - e^{-4KT}\right).$$

The calculations result in the evaluation of the matrix covariance function of the white noise process which drives the system model. This is referred to as the Q matrix and is a nonnegative definite matrix of the form:

$$L = \left(1 - e^{-4KT}\right) \qquad A = b/(b - 2K)$$

$$J = \left(1 - e^{-(2K+b)T}\right) \qquad B = b/(2K - b)$$

$$Y = \left(1 - e^{-2bT}\right)$$

$$Q = q \begin{bmatrix} KA^2L + \frac{8K^2AB}{2K+b} J + \frac{2K^2B^2}{b} Y & KAL + \frac{4K^2B}{2K+b} J \\ KAL + \frac{4K^2B}{2K+b} J & KL \end{bmatrix}$$

APPENDIX E

DESIGN STEPS FOR MULTI-PHASE PROCESSOR

To evaluate the optimal design of the multi-phase array processor, certain data on each processor must be obtained.

1. Cost of each type of processor considered.
2. Time necessary to complete one computation.
3. Power (in watts) used to operate each type processor.
4. Number of packages that compose each processor and number of pins used on each package.

Once the hardware is acquired, the linear program equations are subsequently created. Let C_i = cost of processor P_i, i = 1, 2, 3, . . .

$$C_1P_1 + C_2P_2 + C_3P_3 + \ldots + C_NP_N \leq Z$$

$$C_i \geq C_{i+1},\ i = 1, 2, 3, \ldots N.$$

Let T_c equal total time allowed for matrix computations and T_{pi} equal the cycle time of each processor P_i.

$$T_i = \text{largest integer } (T_c/T_{pi}),\ i = 1, 2, 3, \ldots$$

The time equation will be in the following form:

$$T_1P_1 + T_2P_2 + T_3P_3 + \ldots + T_NP_N \geq \text{(number of terms in matrix)}.$$

Let P_i equal the aggregate of processors of type P_i necessary to compute the problem if only type P_i processors are employed.

$$P' = (\text{number of elements in matrix})/T_1.$$

The resulting linear program is of the form:

Minimize:

$$C_1P_1 + C_2P_2 + C_3P_3 + \ldots + C_NP_N \leq Z$$

Constraints:

$$T_1P_1 + T_2P_2 + T_3P_3 + \ldots + T_NP_N \geq \text{Size}$$

$$W_1P_1 + W_2P_2 + W_3P_3 + \ldots + W_NP_N \leq W_T$$

$$U_1P_1 + U_2P_2 + U_3P_3 + \ldots + U_NP_N \leq U_T.$$

The solution to the linear program will exist in a region bounded above the time line and below the power and area lines. Prior to attempting to obtain the optimal solution, the solution region should be examined to determine if it exists in such a state that will allow the existence of a feasible solution. At this point a reduction or increase of the solution region is achieved by altering the values of W_T and U_T. This capability will facilitate the search for the integer linear program solution by effectively reducing the search domain.

The solution to the integer linear program is generated by using available computer software and computer systems. The technique is to use a branch and bound algorithm based on the Land and Doig (32) method. Details of the algorithm are covered in Appendix A. The end result of the linear program will be a circuit of a practical nature in an optimal form to solve a vector, matrix product computation. Figure 30 illustrates the steps in the design sequence of the Multi-phased array processor.

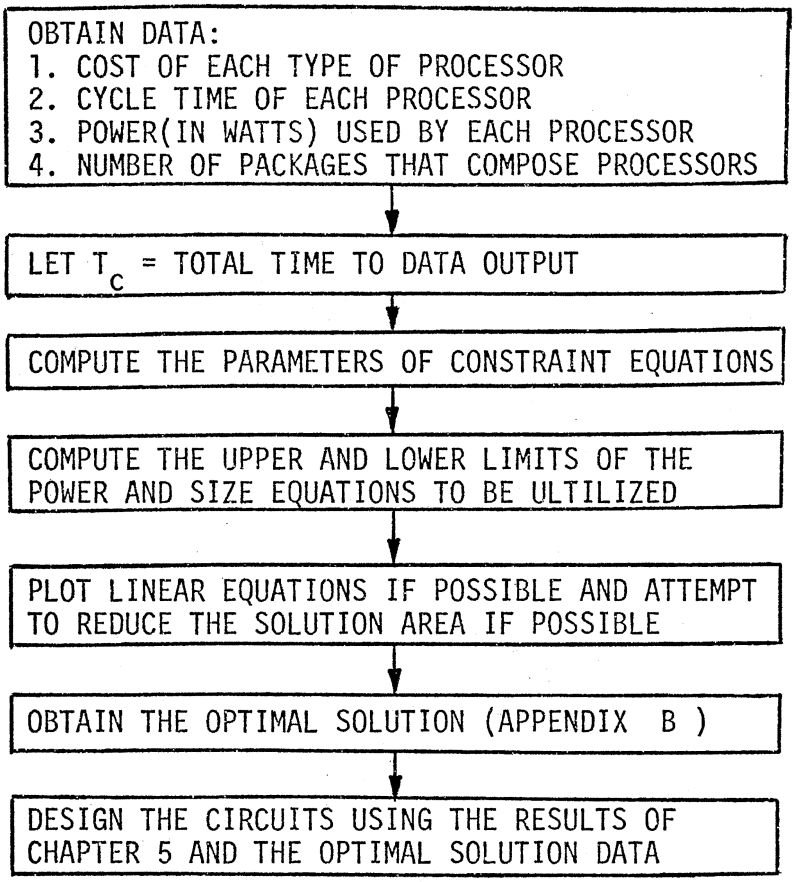


Figure 30. Multi-phased Processor Design Flow Chart

VITA 2

Larry Gene Stotts

Candidate for the Degree of

Doctor of Philosophy

Thesis: OPTIMAL DISTRIBUTED MICROPROCESSOR ARCHITECTURE USING MULTI-PHASE PROCESSING TO PERFORM A VECTOR, MATRIX MULTIPLICATION

Major Field: Electrical Engineering

Biographical:

Personal Data: Born in Pawhuska, Oklahoma, September 7, 1949, the son of Mr. and Mrs. E. E. Stotts.

Education: Graduated from Ponca City High School, Ponca City, Oklahoma, in May, 1967; received the Bachelor of Science degree in Electrical Engineering From Oklahoma State University, Stillwater, Oklahoma, in May, 1972; received the Master of Science degree in Electrical Engineering from Oklahoma State University, Stillwater, Oklahoma, in May, 1977; completed requirements for the Doctor of Philosophy degree at Oklahoma State University, Stillwater, Oklahoma, in July, 1979.

Professional Experience: Communication officer, U.S. Army Signal Corps, May, 1972, to May, 1976; Instructor, Electrical Engineering, Oklahoma State University, Stillwater, Oklahoma, 1978-1979.

Professional Organizations: Member of the Institute of Electrical and Electronic Engineers.