

To the Graduate Council:

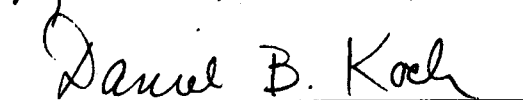
I am submitting herewith a dissertation written by Danny F. Newport entitled "A Methodology for Designing Parallel/Pipelined Systems." I have examined the final copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Electrical Engineering.



D. W. Bouldin, Major Professor

We have read this dissertation
and recommend its acceptance:







Accepted for the Council:



Associate Vice Chancellor and
Dean of The Graduate School

**A METHODOLOGY FOR DESIGNING
PARALLEL/PIPELINED SYSTEMS**

A Dissertation

Presented for the

Doctor Of Philosophy

Degree

The University of Tennessee, Knoxville

Danny F. Newport

December 1995

DEDICATION

This dissertation is dedicated to my wife, Rosemary, and our two children, Sarah and Mac, without whose love, support, and understanding this work might never have been completed. They fill my life with joy and meaning and are my greatest blessing from God.

ACKNOWLEDGEMENTS

In reaching this landmark event, I have received inspiration and support from many individuals and institutions. Words of appreciation are not enough for the efforts, guidance, and patience of my major professor, Dr. Donald Bouldin. He has awaited the closure of this effort with the patience of Job. Special thanks go to Dr. Bob Bodenheimer, Dr. Dan Koch, and Dr. David Straight, the members of my committee, for review of the dissertation and having served on the committee, and in particular, to Bob for his having said, "Danny, you need to finish it up."

A special thanks goes to Christian Michel of the PET center at the Université Catholique de Louvain in Louvain-la-Neuve, Belgium, for the data from my 3D brain scan and the wonderful hospitality during my visit there.

Throughout my graduate work, I have received support and encouragement from the University, Oak Ridge National Laboratory (ORNL), and CTI. In particular, I wish to thank Bill Bryan and Gary Alley at ORNL for their support and encouragement when ASIC design was young in Tennessee.

None of this work would have been possible without the support of CTI. The designs described in this dissertation and the dissertation itself are a direct result of their efforts. There are those at CTI who I feel require a special thank you. Working with John Young on the Crystal Identification and second Coincidence Detection ASICs was a most enjoyable design experience. John was a sounding board for design ideas and did much of the implementation as well as providing the crystal boundary image in Chapter 4. Dr. Ron Nutt and Dr. Mike Casey have continually encouraged and supported my efforts to complete this dissertation. With their support, I have completed what many could

not. I hope that I can serve as an inspiration to all those who have all but dissertation (ABD) completed.

I would like to express my appreciation to my parents and family for their love and support throughout the years. In particular, I want to thank them for allowing me to determine what I did with my life and the foundation they provided that lead to my choosing the right path. My grandfather taught me to walk the path with God and He has greatly blessed me. It is through Him that I am who I am and done all that I have.

There are not enough ways to say thank you to my wife, Rosemary, and our two children, Sarah and Mac, for all of their support. Hence, I have dedicated this work to them. With all my love, I thank you.

ABSTRACT

Serial processing inherently limits the speed at which a given function can be performed. By processing various parts of the function in parallel, the performance can be improved, but at a price - higher design effort. Serial processing lends itself to design methodologies which are very mature, whereas design methodologies for parallel processing are less developed. By pipelining the parallel processing, the performance is again increased, but at an even higher price - design methodologies are even less developed.

This dissertation presents a methodology for designing parallel/pipelined systems based on the design of three systems (two hardware and one software). The methodology assumes that the designer has previous digital design experience and wishes to expand the range of possible designs to include parallel/pipelined implementations. The hardware designs are Application Specific Integrated Circuit (ASIC) designs implemented in CMOS gate arrays. The software design is implemented on an i860 based multiprocessor system. Various architectures for the designs and the corresponding trade-offs are presented including ones which were considered but not used. An emphasis is placed on the application of the methodology to ASIC development, but the broader application to other areas is also presented.

TABLE OF CONTENTS

CHAPTER	PAGE
1. Introduction	1
1.1 Overview	1
1.2 Scope of the Work	3
2. Background	5
2.1 Parallel Processing in General	5
2.2 Parallel Processing Architectures	11
2.3 Parallel Processing Algorithms	15
2.4 Introduction to PET	19
3. Coincidence Detection ASIC	24
3.1 Description of the Problem	24
3.2 Analysis of Possible Architectures	27
3.3 Design and Implementation of First Generation ASIC	38
3.4 Design and Implementation of Second Generation ASIC	41
4. Crystal Identification ASIC	43
4.1 Description of the Problem	43
4.2 Analysis of Possible Architectures	47
4.3 Design and Implementation of ASIC	61

CHAPTER	PAGE
5. 3D Backprojection	69
5.1 Description of the Problem	69
5.2 Analysis of Possible Architectures	73
5.3 Design and Implementation	88
6. Design Methodology for Parallel/Pipelined Systems	99
6.1 Description of the Problem	99
6.2 Analysis of Designs of Parallel/Pipelined Systems	100
6.3 Description of Design Methodology	109
7. Concluding Remarks	114
7.1 Summary	114
7.2 Future Work	115
LIST OF REFERENCES	117
VITA	124

LIST OF TABLES

TABLE	PAGE
2.1 Flynn's classification of computer architectures	13
2.2 Classification of process level architectures	14
3.1 Coincidence detection architectures	34
3.2 Coincidence detection architectural criteria values	35
3.3 Criteria values for $n = 49$, $m = 7$, and $c = 4$	36
4.1 Crystal identification architectures	57
4.2 Crystal identification architectural criteria values	58
4.3 Criteria values for 64 crystals and six-bit input values	59
5.1 Hypercube implementation results	81
5.2 3D backprojection architectures	85
5.3 3D backprojection architectural criteria values	85
5.4 Criteria values for an 8 processor system	86
5.5 3D backprojection execution times for 1 to 12 i860s	91
5.6 3D backprojection speedup and efficiency for 1 to 12 i860s	92
5.7 Sequential fraction for 2 to 12 i860s	96

LIST OF FIGURES

FIGURE	PAGE
2.1 Parallel production lines in a factory	7
2.2 Pipelined production line in a factory	8
2.3 Parallel/pipelined production line in a factory	9
2.4 Flynn's classification of computer architectures	12
2.5 An overview of the positron imaging process	21
2.6 Schematic representation of a commercial PET imaging system	22
3.1 Idealized representation of coincidence function for a PET scanner	26
3.2 Various architectures for coincidence detection	29
3.3 Comparison sequence for a four block cluster	32
3.4 Plot of merit sums versus log of area/power weight ratio	37
3.5 First generation coincidence detection ASIC pipeline	39
4.1 Schematic representation of PET detector processing	44
4.2 Position histogram with crystal boundaries	46
4.3 Sequencer architectures for the analog subsystem	49
4.4 Various architectures for crystal identification	50
4.5 Step-and-test searching for crystal identification	52
4.6 Non-comparator, RAM-based energy validation architectures	56
4.7 Plot of merit sums versus log of area/power weight ratio	60
4.8 Block diagram of crystal identification ASIC	63
4.9 Crystal identification ASIC floorplan	65
5.1 2D coincidence detection configuration	70

FIGURE	PAGE
5.2 3D coincidence detection configuration	71
5.3 Multiprocessor system for 3D backprojection	74
5.4 SPSP architectures for 3D backprojection	76
5.5 Parallel/pipelined architectures for 3D backprojection	78
5.6 Transputer architectures for 3D backprojection	82
5.7 Plot of merit sums versus log of area/performance weight ratio	87
5.8 3D backprojection state diagram	90
5.9 Speedup of 3D backprojection versus number of processors	93
5.10 Efficiency of 3D backprojection versus number of processors	94
5.11 Single image slice from the author's 3D brain scan	98
6.1 Levels of system design	101
6.2 Coincidence detection merit sums versus log of area/power weight ratio	106
6.3 Parallel/pipelined design methodology	110
6.4 Floorplan for crystal identification ASIC	113

CHAPTER 1

Introduction

1.1 Overview

Parallel systems in electronics and computer processing have been receiving an increasing amount of attention in recent years. This is due to the ever present "problem waiting for a solution" - as processing power/speed increases, the problems to be solved become larger. As an example, entering commands to a computer through a keyboard requires very little processing power, but significantly more processing power, at present, is required when the desire is for the computer to recognize spoken commands.

Serial processing inherently limits the speed at which a given function can be performed. However, serial processing lends itself to a straightforward design methodology - the processing steps are defined and each can be optimized separately using well established techniques. Note that there are other design methodologies for serial processing in which either the entire design or different parts of the design are optimized together. The optimizations are based on what the designer deems most important (speed, power, size, cost, etc.). These later methodologies are under continuing research efforts involving multi-dimensional optimization techniques [19].

By processing various parts of the function in parallel, the performance can be improved, but at a price - higher design effort. Although design methodologies for parallel processing do appear less developed, the concept of parallel or concurrent processing has been investigated and utilized since the 1950's and 60's [17, 21, 23]. However, much

of the effort has been in the design and implementation of programming languages for concurrent processing [65]. Much effort has also gone into the design of multiprocessor systems. Specific parallel processing architectures for Very Large Scale Integration (VLSI) have also been developed, particularly for certain applications (i.e. image processing [15]).

By pipelining the parallel processing, the performance is again increased, but at an even higher price - design methodologies are even less developed [19]. With higher density field-programmable gate arrays (FPGA) and Application Specific Integrated Circuits (ASIC) available, parallel/pipelined processing can be implemented by a much larger group of experienced designers than previously possible. As these designers begin a design for a specific application, a set of questions should be asked:

Can this specific application be designed as a parallel/pipelined system?

Will a parallel/pipelined system for this specific application be
"better" than a serial design?

How do you design a parallel/pipelined system for this specific application?

Are the hardware/software resources available for a parallel/pipelined
system for this specific application?

Will the design time/effort for a parallel/pipelined system for this
application be greater/less than a serial design?

By having a design methodology, these questions can be more readily addressed by the designer earlier in the design cycle.

1.2 Scope of the Work

The scope of this dissertation is to identify a design methodology for parallel/pipelined systems with an emphasis on ASIC development. This methodology answers the questions posed above. The methodology assumes that the designer has previous digital design experience and wishes to expand the range of possible designs to include parallel/pipelined implementations.

First, a review of the general problem of parallel processing and its application to diverse areas are presented in Chapter 2. These areas include production lines, traffic flow, communications, product design, computer operating systems, multiprocessor systems and software, and digital and analog electronics. The three systems used in this dissertation to derive the design methodology are all subsystems used in a commercial Positron Emission Tomography (PET) medical imaging system. Therefore, a brief overview of the PET imaging system is presented.

Chapters 3 through 5 present three design problems which have been designed and implemented as parallel/pipelined systems. The coincidence detection problem is posed in Chapter 3 with an analysis of possible architectures. Two parallel/pipelined ASIC designs and implementations are presented. The two designs differ due to a change in the system requirements and technology improvements.

In Chapter 4, the crystal identification problem is posed with an analysis of possible architectures. A parallel/pipelined ASIC design and implementation are presented.

The 3D Backprojection problem is posed in Chapter 5 with an analysis of possible architectures for implementation on a multiprocessor system. A parallel/pipelined design and implementation is presented for an i860 based multiprocessor system.

Based on an analysis of the designs presented in Chapters 3 through 5, an empirically derived design methodology for parallel/pipelined systems is presented in chapter 6. The design methodology is emphasized for ASIC development and experienced designers, but its application to other areas is also discussed. A summarization and future work conclude the dissertation.

CHAPTER 2

Background

2.1 Parallel Processing in General

To some, efficient parallel processing, be it on a multiprocessor system or in an ASIC, appears to be more of an art than a science. In some ways, this may be true. However, effective parallel processing existed long before the advent of the computer or the integrated circuit. Farmers have long practiced a form of parallel processing by either planting multiple crops in the same field at the same time or planting different crops throughout the year in different fields. The author, while growing up, had first-hand experience of planting corn and green beans in the same field. This allowed the two crops to be planted and cared for simultaneously which halved the work. The corn also served as poles for the green beans to "climb", thus eliminating the task of putting poles in place. Having multiple workers plant, weed, and harvest a single crop is parallel processing which has occurred since the first field was plowed.

When an individual gets into his/her car and drives on public highways, a form of parallel processing is at work. Each driver is processing and acting on information from other drivers to, hopefully, drive safely and allow traffic to flow smoothly. A traffic jam occurs when there is a breakdown in this processing or the system becomes overloaded.

Production lines in factories are perhaps the most obvious and overworked examples of parallel processing in the non-computer world. But they provide an easily described

and understood parallel processing system which people can relate to. Thus production lines will be used as a basis for a description of parallel processing systems.

Production lines can be classified into one of three categories: parallel, pipelined, or parallel/pipelined. Parallel production lines have each worker doing all the tasks necessary to produce the finished product. An example of this type of production line is in the clothing industry. Each worker is given all the pieces for a shirt and sews them together to produce a shirt. Figure 2.1 illustrates parallel production lines in a factory.

Pipelined production lines have each worker doing a specific task with the product proceeding from worker to worker until it is finished. When the worker completes his/her task on one item and passes it to the next worker, another is given to him by the worker before him. An example of this type of production line is in the automobile industry. Each worker performs a specific task, such as attaching the steering wheel or the dash, and the partially completed car is moved from worker to worker until it is finished. Figure 2.2 illustrates a pipelined production line in a factory.

A parallel/pipelined production line combines the features of both of the above techniques. An example of this type of production line can also be found in the automobile industry. Each worker performs a specific task, but there may be several workers performing the same or different tasks on the *same* car at the *same* time and the partially completed car is moved from stage to stage until it is finished. For example, one worker may attach the steering wheel at one stage and four workers may each attach a tire at the same time at the next stage. Thus, balancing the time required at each stage to improve some measure of performance. Figure 2.3 illustrates a parallel/pipelined production line in a factory.

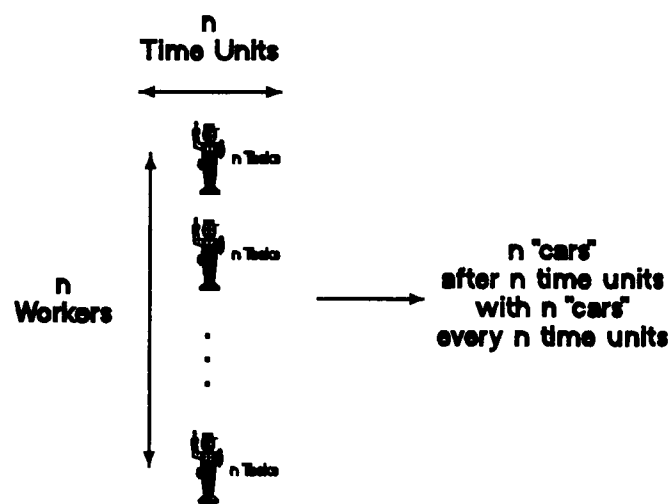


Figure 2.1. Parallel production lines in a factory.

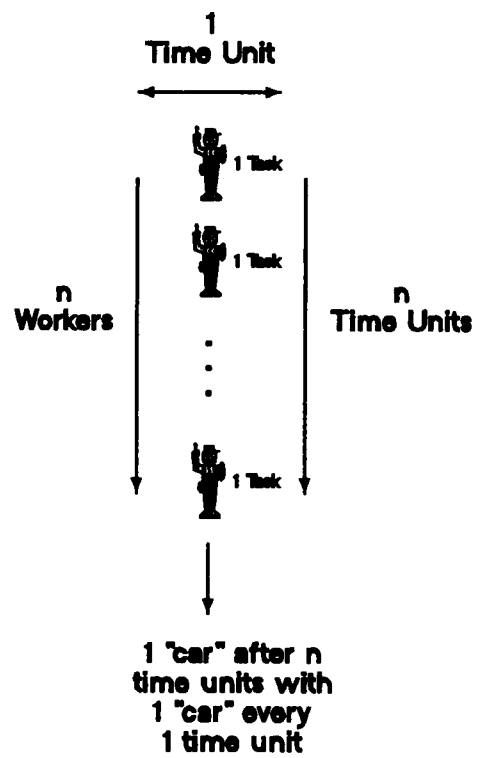


Figure 2.2. Pipelined production line in a factory.

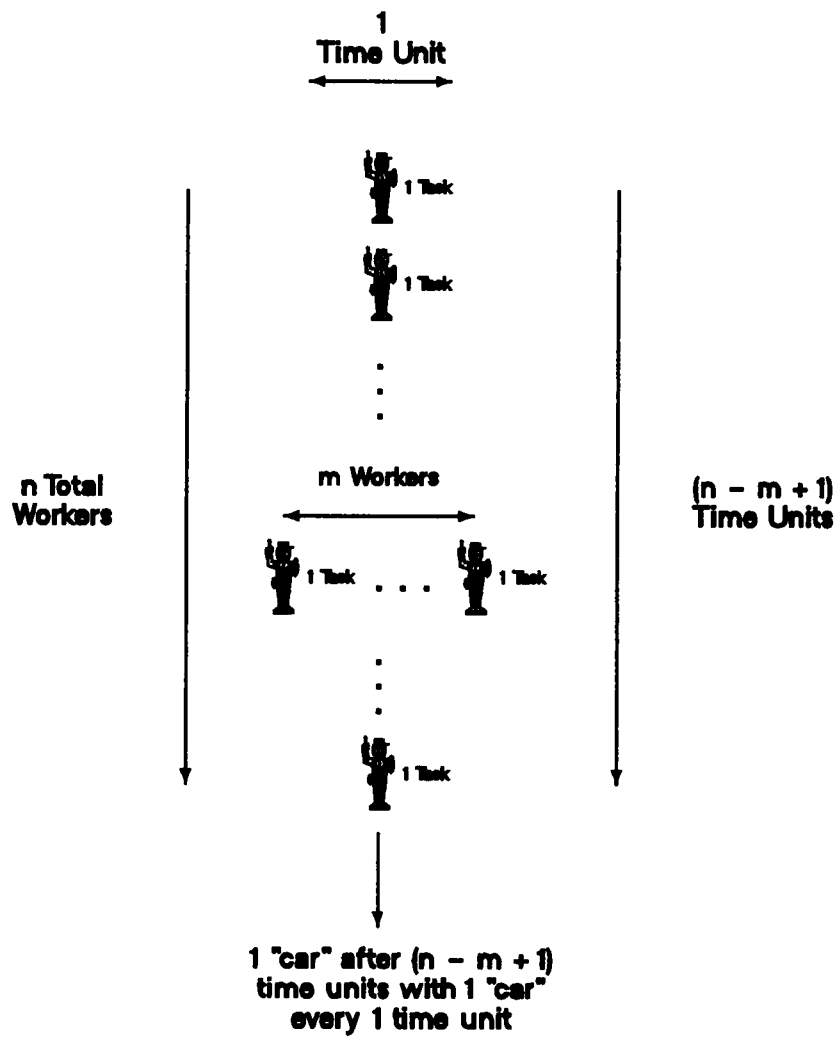


Figure 2.3. Parallel/pipelined production line in a factory.

The three categories presented above exhibit different properties of parallelism. Hwang [34] refers to the parallelism exhibited by parallel production lines as *spatial parallelism*. Specifically, the parallelism is distributed across space, i.e. multiple workers perform the same (or different) task to produce multiple results. Hwang refers to pipelined processing as *temporal parallelism*. Specifically, the parallelism is distributed across time, i.e. multiple workers perform different tasks at different times to yield a single result multiple times. Parallel/pipelined processing exhibits both *spatial* and *temporal parallelism*. Specifically, the parallelism is distributed across both space and time as required.

To illustrate the differences in the categories, an example using an automobile factory can be used. Assume that there are n workers available, that it takes n tasks to build a car, and that each task takes the same amount of time, 1 time unit (an *ideal* situation). Thus a single worker doing every task can build a car in n time units.

There are two possible forms of spatially parallel production line. One would have each worker doing every task to build their own car. At the end of n time units, n cars would be produced and another n cars would be produced at the end of every n time units. If it were possible for all the tasks to be performed at once, each worker could perform a different task on the same car. Thus at the end of 1 time unit, a single car would be produced and another car would be produced every time unit after that.

A temporally parallel production line would have each worker doing a different task, one after the other, to build a single car. All the workers stay busy by working on different cars at different times. Thus after n time units, a single car is produced, but another car is produced every time unit after that. If certain tasks in building a car can be performed at the same time, such as the steering wheel and all four tires installed,

a spatially/temporally parallel production line would have multiple workers doing the same (or different) task at the same time on the same car. Assume that m tasks are performed at one stage of the production. After only $n - m + 1$ time units, a single car is produced, but another car is produced every time unit after that. Taken to the extreme, if all the tasks could be performed at once with each worker performing a single task, after only 1 time unit, a single car would be produced with another car produced every time unit after that. This is exactly the same result as with the second form of the spatially parallel production line. Thus the parallel/pipelined category bridges the void between the other categories.

2.2 Parallel Processing Architectures

Production lines provide a simple and easily understood example of parallel processing, but the goal is a methodology for designing parallel/pipelined ASIC systems. Toward this end, relating parallel/pipelined ASIC designs to multiprocessor systems, where a great deal of work has been done, would be of great benefit. Thus, an understanding of multiprocessor systems is in order.

In 1972, Michael Flynn [24] classified parallel computer architectures into categories based on instruction and data streams. This classification is still widely used today. Table 2.1 lists Flynn's classification and Figure 2.4 shows the various architectures.

Conventional von Neumann computers are *single instruction over single data stream* (SISD) architectures, Figure 2.4a. These architectures execute a single instruction on a single set of data and proceed to the next instruction and data set. Although one might think that most modern microprocessors are SISD computers, they are not. Most

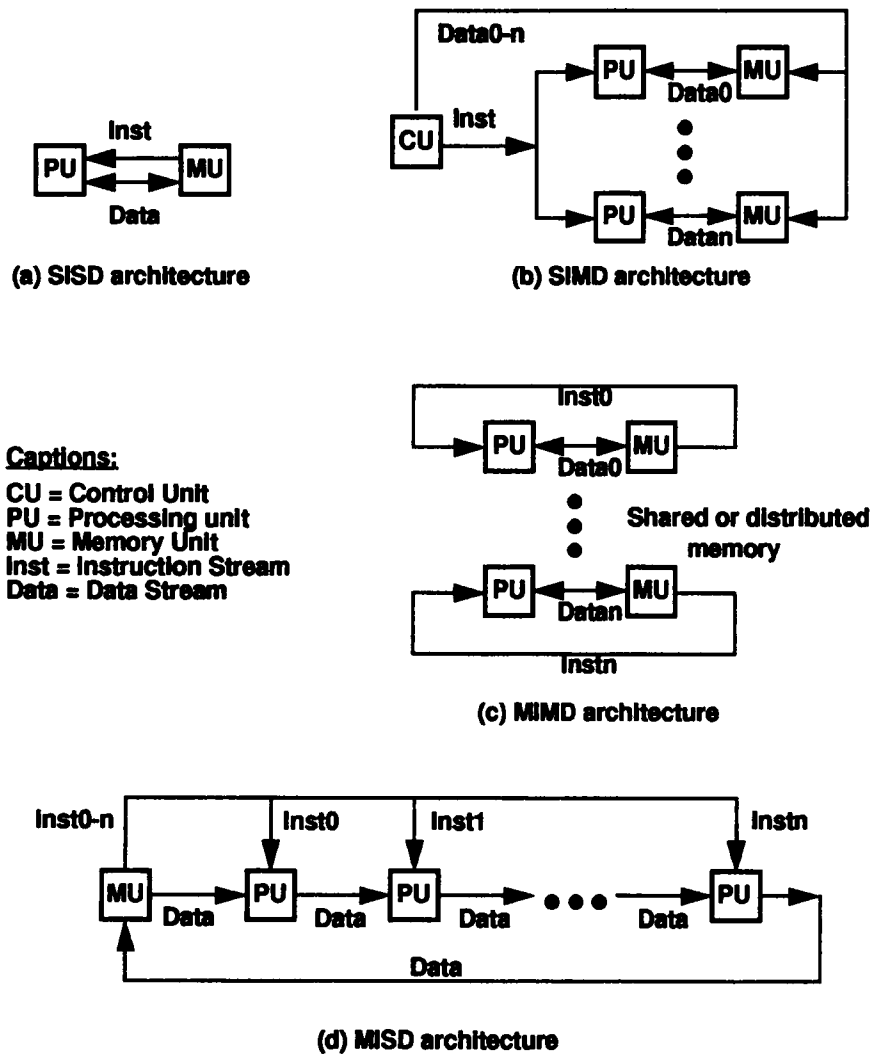


Figure 2.4. Flynn's classification of computer architectures (Adapted from [34]).

Table 2.1. Flynn's classification of computer architectures.

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

modern microprocessors perform some internal form of parallel processing and are not true SISD architectures.

Figure 2.4b shows a *single instruction over multiple data streams* (SIMD) architecture. In these architectures, the processors execute the same instruction on different data sets. If the instruction is to add the values A and B , each processor would add its own values of A and B . Array processors are an example of this architecture.

A *multiple instructions over multiple data streams* (MIMD) architecture is shown in Figure 2.4c. The processors in this architecture execute different instructions on different data sets. Hwang [34] states that most parallel computers for general purpose computations have been built using this architecture. The hypercube, iPSC/2, is an example of this architecture.

Systolic arrays [45, 44] tend to be classified as *multiple instructions over single data stream* (MISD) architectures, Figure 2.4d. The processors execute different instructions as the same data stream is passed from processor to processor. This architecture is particularly suited for image applications and has produced excellent results [66, 15].

The above architectures describe processors and their *instruction* level processing of data streams. One can broaden this definition to architectures concerning *processes*, i.e. connected sets of operations, on data streams in the processors. Thus, one can

Table 2.2. Classification of process level architectures.

	Single Process	Multiple Process
Single Data	SPSD	MPSD
Multiple Data	SPMD	MPMD

describe process level parallel architectures [34]. These architectures follow the same form as the above architectures and can be classified as in Table 2.2.

A personal computer running DOS is an example of a *single process on single data stream* (SPSD) architecture. In hardware, the traditional form of a state machine is a SPSD architecture - a given state (single process) with a given set of inputs (single data) produces a single next state.

An array processor where each processor executes the same set of operations on different sets of data is an example of a *single process on multiple data streams* (SPMD) architecture. A multitasking operating system such as UNIX is an example of a *multiple processes on multiple data streams* (MPMD) architecture. Finally, a systolic array in which each processor performs a different set of operations instead of a single instruction is an example of a *multiple processes on single data stream* (MPSD) architecture.

There are other aspects of multiprocessor systems which should be described. These are: grain size, interconnection, and coupling [43]. The grain size, or *granularity*, of a multiprocessor system is defined by the capability and number of processors in the system. A *coarse-grained* system has tens of processors usually complex with large memories. A *medium-grained* system has hundreds of processors usually less complex

with smaller memories. A *fine*-grained system has thousands of processors usually simple with very small memories.

The interconnection, or *topology*, of multiprocessor system is the communication network which connects the processors. Various interconnection schemes are rings, trees, cubes, and meshes. The coupling of a multiprocessor system is defined as either *tight* or *loose*. Processors which share memory or the same clock form a tightly coupled multiprocessor system. A loosely coupled multiprocessor system has processors which do not share memory (distributed memory) or the same clock.

2.3 Parallel Processing Algorithms

Most computational algorithms have been designed to execute on SPSD architectures. These algorithms do not necessarily map well to parallel architectures. Therefore, algorithms for parallel processing should be designed specifically for parallel architectures. Due to architectural and overhead limitations, the algorithms must be designed for the specific architecture on which they are to run to be efficient [34].

There is no explicit methodology for designing an algorithm for a specific problem. However, there are some strategies to consider when designing an algorithm [43]. These include: divide-and-conquer, search methods, algebraic transformations, random trial methods, and approximation methods.

The divide-and-conquer strategy breaks the problem into many smaller problems which are distributed to the processors in a multiprocessor system. Care must be taken in the distribution of the smaller problems to balance the workload of each processor.

The specific problem may require a combining of the results from all the processors to complete the problem.

Search methods are applied when one is trying to find the "optimal" solution to a problem. A "cost" function is defined for the problem and each processor is given a different set of points to begin the search. Various approaches to searching the solution space are available and can be found in [43].

Algebraic transformation methods are ones in which the problem is transformed from one domain into another to be solved. The final solution is obtained by inverse transforming the result back into the original domain. An example of this is a problem involving convolution in the time domain. The problem is transformed into the frequency domain using the Fast Fourier Transform and a multiplication is performed. The result is inverse transformed to yield the desired solution.

Methods which use random numbers for searching or estimation are considered random trial methods. These methods are useful for "near optimal" solutions of NP-complete problems such as the traveling salesman problem which has a factorially large search space. Monte Carlo, simulated annealing, and genetic algorithms are typical random trial methods.

Finally, approximation methods are used to produce solutions which are not exact. However, the computation time required is much less than using an exact method. The numerical results must still be reasonably close to the exact solution for the method to be considered useful. Evaluation of a function by a Chebyshev polynomial is an example of an approximation method.

Parallel algorithms which are implementable on real processors have other important characteristics. Among these are: deterministic, computational granularity, parallelism

profile, communication patterns and synchronization requirements, uniformity of operations, and memory requirements and data structures [34].

Deterministic algorithms have each operational step uniquely defined. This is the way real computers execute programs. Computational granularity is similar to the granularity defined above for multiprocessor systems. But, the level of computation granularity is based on the size of data items and program modules instead of the number and complexity of processors. Also, most algorithms have an inherent portion of their execution which is sequential which becomes dominant as more processing power is applied to the parallel portion. This is defined as Amdahl's Law [34].

The effectiveness of a parallel algorithm can often be determined by doing a parallelism profile of the algorithm. The parallelism profile of an algorithm is the distribution of the degree of parallelism in an algorithm. The main goal is to determine where there is an opportunity for parallel processing.

Most overhead in parallel processing can be attributed to communications and synchronization. Thus, determining the communication patterns and synchronization requirements for an algorithm will yield an insight into the overhead costs associated with the algorithm. Communication patterns can be either static or dynamic, i.e. not changing or changing with time.

The types of operations required by an algorithm on a data set often determine the suitability of an algorithm for a specific multiprocessor architecture. A uniform set of operations on a data set may run well on all the architectures, but a random set of operations is better suited for a MPMD multiprocessor architecture.

Finally, one of the major concerns in parallel processing is the memory requirements of an algorithm. A determination of the memory requirements of an algorithm will

yield valuable insight into the suitability of an algorithm for a particular multiprocessor system. In short, the question to ask is: "Is the memory required by this algorithm available on this multiprocessor system?" This will also aid in determining the choice of a shared or distributed memory structure.

When comparing two or more objects, one needs a common reference to fairly describe their differences and similarities. For multiprocessor systems and algorithms, several performance measures are available. Two of these measures, speedup factor and efficiency, are the ones typically used. Other measures and their usage may be found in Hwang [34].

The speedup factor for an n -processor system is defined by Hwang as

$$S(n) = T(1)/T(n) \quad (2.1)$$

where $S(n)$ is the speedup factor, n is the number of processors, $T(1)$ is the execution time on one processor, and $T(n)$ is the execution time on n processors.

The efficiency is for an n -processor system defined by Hwang as

$$E(n) = \frac{S(n)}{n} = \frac{T(1)}{nT(n)} \quad (2.2)$$

where $E(n)$ is the efficiency, $S(n)$ is the speedup factor, n is the number of processors, $T(1)$ is the execution time on one processor, and $T(n)$ is the execution time on n processors.

The speedup can be related to the sequential portion of the program and the number of processors by [34]

$$S(n) = \frac{n}{1 + (n-1)\alpha} \quad (2.3)$$

where $S(n)$ is the speedup, n is the number of processors, and α is the sequential fraction of the program.

2.4 Introduction to PET

The three systems used in this dissertation to derive the design methodology are all subsystems used in a commercial Positron Emission Tomography (PET) medical imaging system. Therefore, a brief overview of the PET imaging system is warranted to familiarize the reader with any terminology or system details. A further description of PET and its applications can be found in [59].

In a PET imaging system, a patient is injected with a radioactively tagged substance which the body normally metabolizes in some fashion. The radioactive tag used is a positron emitting isotope of either an element found in the substance or an element which is substituted for another element in the substance. For example, one of the most widely used isotopes is the positron emitting isotope of fluorine, ^{18}F . ^{18}F can be substituted, through a chemical synthesis process, for hydrogen in complex compounds such as glucose forming Fluro-deoxyglucose (FDG). When FDG is injected into a patient, the body will attempt to use it in the same fashion as it would normal glucose. Thus, there will be higher concentrations of positron emitters in areas where glucose is metabolized at higher levels, namely the brain, muscle tissue (the heart), and tumors.

When a positron encounters an electron, both are destroyed due to the positron being the antimatter equivalent to the electron. The energy released from their destruction is in the form of two gamma rays which are traveling in opposite directions (i.e. the angular difference is 180°). Gamma ray detectors encircling the patient detect the gamma rays, one on either side of the patient, and the time at which they were detected. Therefore, when two detectors on opposite sides of the patient have detected gamma rays that

occurred within some time window of each other, it is safe to assume that the positron-electron interaction occurred somewhere along the line connecting the two detectors.

By histogramming these occurrences based on these lines of response (LOR), a pattern which uniquely describes the distribution of radioactivity is formed. The array in which the histogram is formed is typically called a sinogram and can be thought of as the Radon transform of the isotope distribution. An image of the isotope distribution can therefore be formed by taking the inverse Radon transform of the sinogram. Note that in a "real" system there are corrections which must be applied to the data due to limitations of the detector system, but which are disregarded here for convenience.

The inverse Radon transform is typically calculated using filtered backprojection. Simply, each row of the sinogram, which corresponds to different collection angles, is filtered with a highpass filter. The contribution of each row to a given pixel is determined by projecting a line from the pixel to the filtered row. The total contributions (sum) from all rows (angles) form the pixel values in the resulting image and correspond to the isotope distribution in the patient. Further descriptions of filtered backprojection can be found in [32, 29]. Figure 2.5 shows an overview of the above process.

A schematic representation of a commercial PET imaging system (CTI, Knoxville, TN) is shown in Figure 2.6. The three designs used in this dissertation are subsystems within this system. The processing occurs as follows. The Clock Generator produces a master clock from which all the Detector Electronics and Coincidence Processor derive their timing. Gamma rays are detected by blocks of bismuth germanate (BGO) crystals coupled to photomultiplier tubes. The signals from the photomultiplier tubes are processed by a set of analog electronics which determine the X and Y positions of the gamma ray event within the block, the energy of the event, and the time of the event.

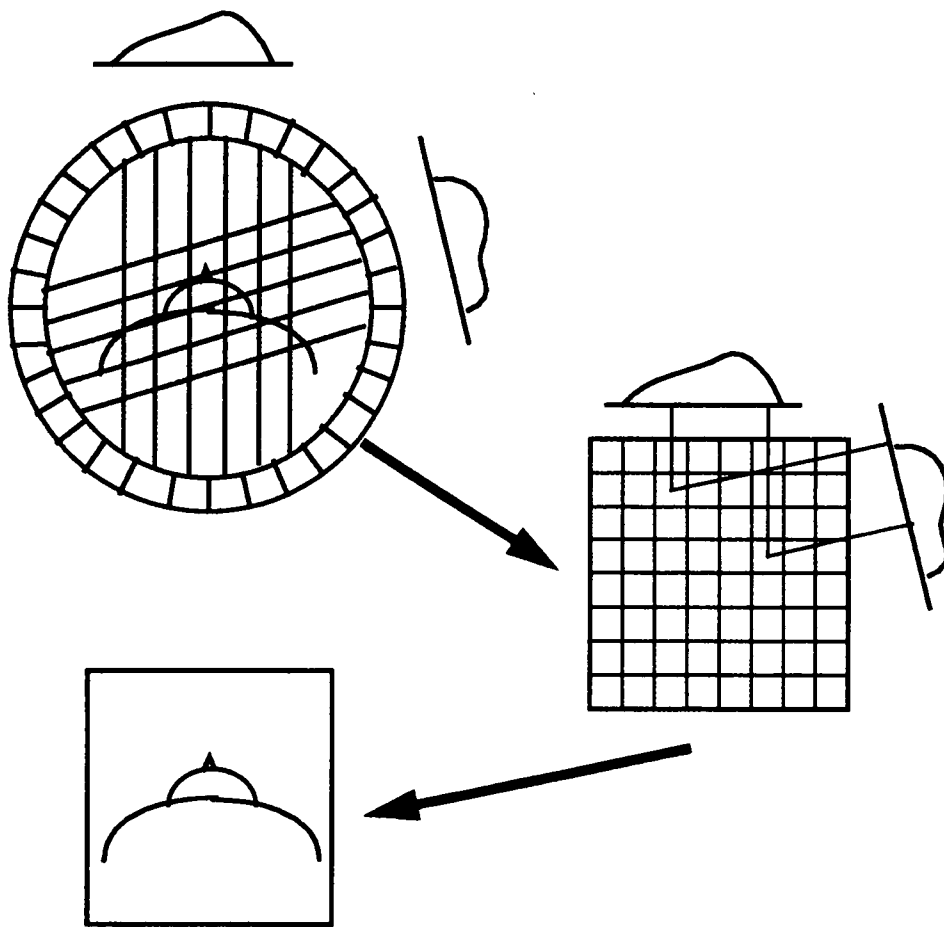


Figure 2.5. An overview of the positron imaging process (Adapted from [8]).

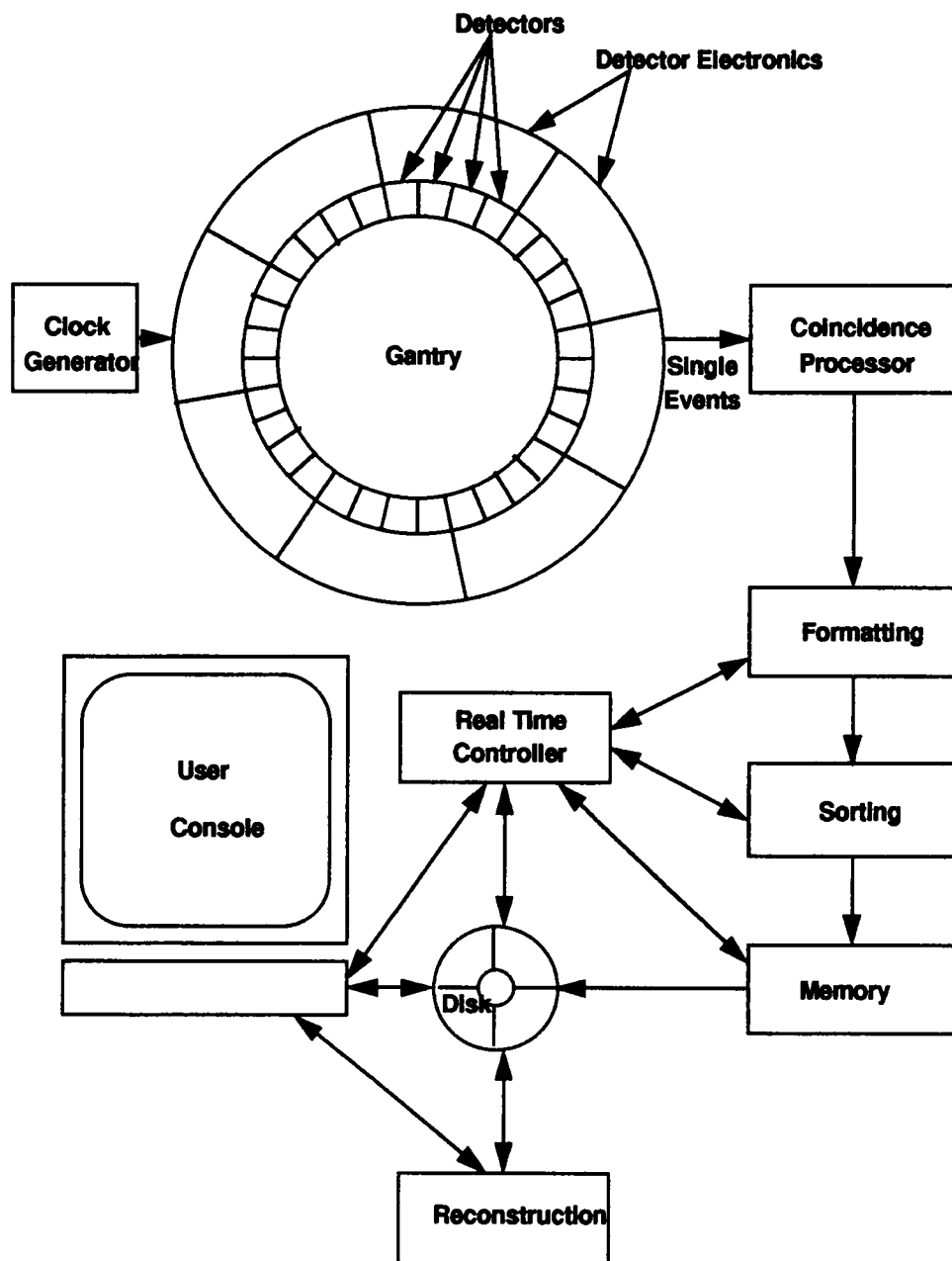


Figure 2.6. Schematic representation of a commercial PET imaging system (Adapted from [8]).

The X and Y positions are used to determine the detector (crystal) in which the event occurred and the energy is used to validate the event. The time of the event is relative to a 256ns cycle time. The Crystal Identification ASIC performs these functions.

Valid events are transmitted to the Coincidence Processor as serial bit streams synchronized to the master clock. The data transmitted includes the event position (crystal number and position in the gantry) and the time the event occurred within the 256ns cycle. The Coincidence Processor receives input from all positions around the Gantry. All events received by the Coincidence Processor are compared to determine which events are time coincident for a given 256ns cycle. The Coincidence Detection ASIC performs this function in the Coincidence Processor.

Time coincident events are transmitted to the Formatting and Sorting for histogramming. The Coincidence Processor transmits the event positions which had a time coincident and the Formatting translates this into an LOR. The Sorting uses this LOR to histogram the event in memory. When a study has been completed, the sinogram is stored onto a disk so that Reconstruction may be performed. The Reconstruction uses the 3D Backprojection software. The 3D Backprojection software uses an extension of the filtered backprojection presented in the overview above for true 3D reconstruction. The image of the isotope distribution is displayed on the User Console after reconstruction has been completed.

CHAPTER 3

Coincidence Detection ASIC

3.1 Description of the Problem

As described in Chapter 2, coincidence detection involves taking time inputs from multiple sources and comparing them to determine which, if any, are within a specified time window of each other. Coincidence detection occurs not only in PET but in other areas such as high energy physics and nuclear instrumentation. In these areas, coincidence detection has traditionally been performed using pulses where the pulse is generated when an event occurs and the width of the pulse is the time window desired. These pulses are used as inputs to an array of AND gates with SET-RESET flip-flops capturing the coincident events. There are problems with this type of system. The pulse timing must be accurate in both the generation of and width of the pulses. Skew between pulses at the input of the AND gates must be minimal and the flip-flops must be able to handle narrow pulses. However, many systems of this type have been built and function very well.

For the system described here, this "analog" form of coincidence detection is not used. When an event occurs, the time of the event, relative to a 256ns cycle, is generated as a digital value (6 bits). Therefore, a coincident event can be found by comparing two digital values and determining if they are within a specified range. The critical aspect of this approach is that the generation of the digital time values must be matched throughout the system.

An idealized representation of the coincidence detection function for a PET scanner is shown in Figure 3.1. The inputs are divided into two groups labeled "A" and "B." A single "A" input is compared with a specific set of one or more "B" inputs. These "A/B" comparison points are called "module pairs" for identification.

The digital time values and the position of the single events are transmitted serially from all positions on the gantry. The serial transmissions are synchronized to a master clock. Each data stream is eight bits long with each bit being 32ns wide, thus the 256ns cycle. Therefore, the coincidence detection system has a cycle time of 256ns with a subcycle of 32ns for each serial bit.

There is no requirement that the coincidences be available on the next cycle after the inputs have arrived. As a matter of fact once the coincidences have been determined, the transmission of the coincidences to the next subsystem can be out of time sequence. Note that there are some PET studies which are "timed," but the timing resolution of these studies is on the order of milliseconds. Therefore, the coincidence does need to be transmitted to the next subsystem within a few milliseconds of when it occurred.

Another requirement of the coincidence detection is that the "planes" in which the two single events occurred be within a certain range of each other. The plane in which a single event occurred is essentially the axial position of the detector on the scanner. Therefore, a valid coincidence must be both time and position coincident. The position information is a digital value and, thus, can use the same type of comparison logic as the time comparison.

Finally, the requirements on coincidence detection and output of these events changed between the first generation ASIC and the second. This was due to a better understanding of the overall PET environment during the design of the second generation ASIC

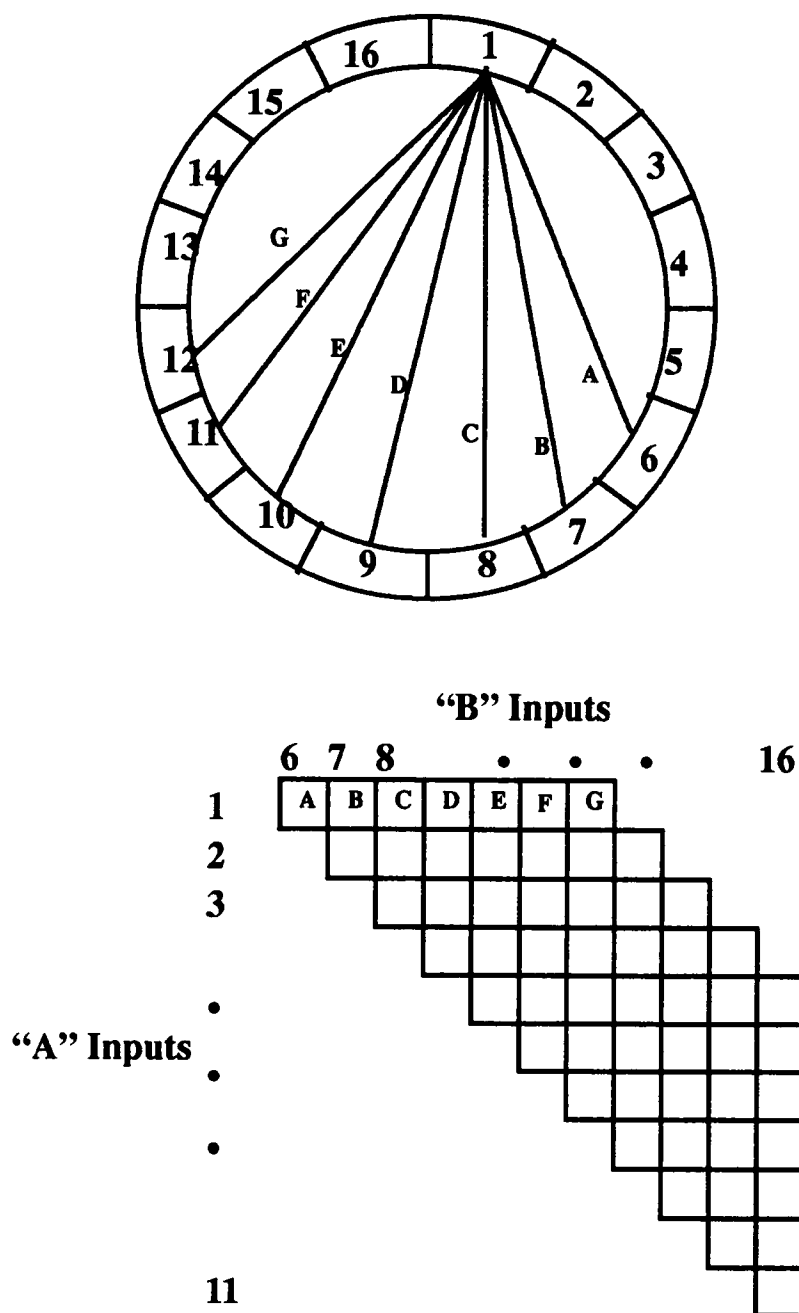


Figure 3.1. Idealized representation of coincidence function for a PET scanner.

and a desire for improvements. In the first generation ASIC, the requirement was that a coincidence occurred when the two time values were within ± 1 of each other. In the second, the requirement was that the time window be selectable (i.e. 0, ± 1 , ± 2 , etc.). Also, in the first generation ASIC, the coincidence detection was driven by a concept known as a "multiple." A multiple is defined as a single event being time coincident with two or more other single events. This requires knowing what other coincidences have occurred within this cycle and identifying those which are multiples. The requirement of identifying multiples was eliminated in the second generation ASIC.

In the first generation ASIC, the requirement appeared to be that only one coincidence from the current cycle could be output. By eliminating the concept of the multiple, it was realized that events could be buffered and output on the next available cycle. This allowed the second generation ASIC to have far better performance [70].

Some additional requirements which had very minor system impact have not been presented here for convenience. Further details can be found in [22, 56, 70].

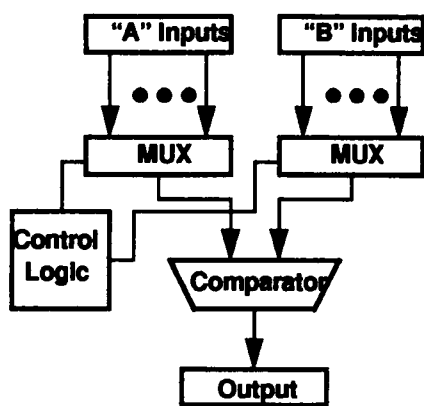
3.2 Analysis of Possible Architectures

Using the above problem description (or, loosely interpreted, design specifications), possible architectures may be analyzed. With no input/output timing relationship required (relatively speaking the output based on a given input must occur within x cycles - for the above, about 10,000 cycles - a *virtual* eternity in digital design), pipelining may be utilized. From the description, a three stage pipeline is the minimum (input, coincidence detection, output) and about 10,000 is the maximum. Taking the minimum

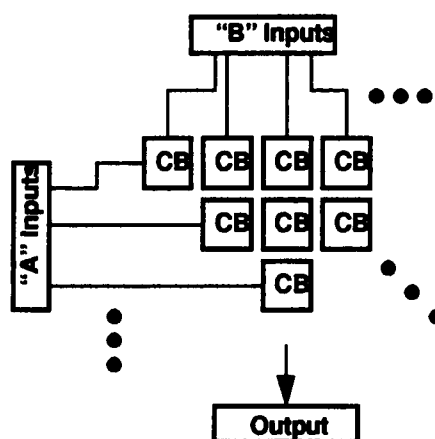
three stage pipeline identified, each stage of processing can be considered separately. If additional stages of pipelining are found to be needed, it is easily added.

The input and output stages are very straightforward. Both may possibly perform two functions: buffering and format conversion. Buffering means holding the data until it is to be used. For the input, this may mean additional pipeline stages until a particular input is used in the coincidence detection process or is passed on to the output. For the output, this may again mean additional pipeline stages until it is time to output the data. Format conversion essentially means from serial to parallel or from parallel to serial depending upon the needs of the next stage.

The main stage to be analyzed is the coincidence detection where the actual processing is taking place. Various approaches to coincidence detection can be defined and are shown in Figure 3.2. Figure 3.2a shows a uniprocessor (SISD) approach to coincidence detection which *loops* through the inputs and performs the comparisons. For this architecture, the time available for each comparison is equal to the number of comparisons required during one cycle divided by the cycle time. For a small number of comparisons, this architecture has a size (spatial) advantage due to multiplexors being relatively small in VLSI terms compared to the size of a comparison block. However to maintain the necessary throughput as the number of comparisons increases, the available time to do the comparison will decrease. At a detailed level, switching and settling times for the multiplexors and control logic must be included in the overall comparison time. For architectural analysis, an estimate of the comparison time which includes these times can be used. Power consumption is also an issue with this architecture. As the number of comparisons increases, the power consumption increases due to the comparison cycle



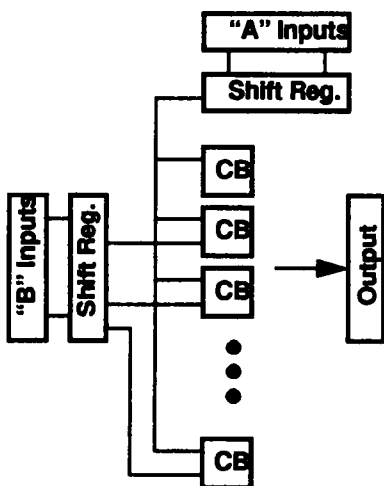
(a) SISD architecture



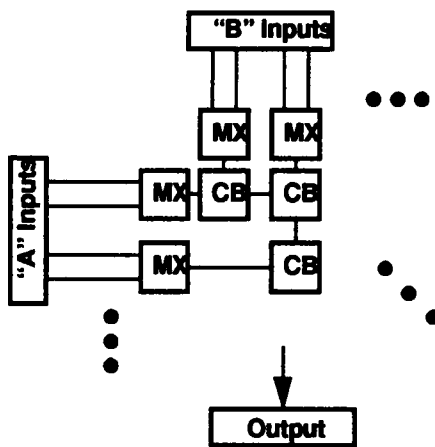
(b) SIMD architecture

Captions:

CB = Comparison Block
MX = Multiplexor



(c) "Shift and compare" architecture



(d) "Cluster" architecture

Figure 3.2. Various architectures for coincidence detection.

time decreasing. References to power consumption are based on implementations in CMOS where the power is directly proportional to the frequency of operation.

Regardless of which generation ASIC is considered, the coincidence detection is a comparison between multiple sets of data. Thus, a *single-instruction, multiple-data* (SIMD) type architecture is possible, Figure 3.2b. For this architecture, the comparison has a full cycle time to complete, but there must be a comparison block for each module pair. Thus it has a temporal advantage, but a spatial disadvantage. However, it may be possible to use this temporal advantage to produce a size reduction. In particular, a serial comparison of the data may be performed instead of a parallel comparison (which has been assumed in these discussions). The trade-off then becomes one of power consumption. An eight bit parallel comparison block running at a cycle time of 256ns will consume far less power than a serial comparison block running at a cycle time of 32ns. The reason for this is the clocking of the flip-flops holding the propagation data. If there are two bits of propagation data, the above serial comparison block will consume about 16 times the power as the parallel comparison block, but it will be about 1/6 to 1/8 the size.

Two other architectures can be identified which are hybrids of the two architectures above. Both architectures combine multiple comparison blocks with multiple data sets over time to form a multiple SISD architecture, in essence a SIMD architecture with a SISD subsystem. The difference between the two is in how data gets to the comparison blocks. Figure 3.2c is a “shift and compare” architecture [52] which has a row of comparison blocks. The data for the comparison blocks is from input data that is shifted horizontally and vertically at each subcycle of the coincidence detection cycle time. The number of comparison blocks is reduced to the number of “column” comparisons, but

the number of subcycles must equal the number of "row" comparisons. This produces a problem similar to that with the SISD architecture: as the number of "row" comparisons increase, the time available for the comparison decreases. Power is also an issue with this architecture since all of the input data must be shifted at each subcycle. Again, as the number of subcycles increases, the power consumption increases by a factor of two.

Figure 3.2d shows the second hybrid architecture. In this architecture, the comparisons are performed on local clusters of data. The data is multiplexed into the comparison blocks. From a VLSI viewpoint, the size of the clusters is influenced by the time needed for a comparison and the size of a comparison block. The clusters are rectangular regions within the array. By this definition, the first hybrid architecture can be viewed as a form of this architecture with the cluster being a "row." Having the clusters more localized, i.e. more square, should reduce any communications overhead. But, the "row" cluster has the advantage of being a straightforward design with the shift registers performing the brunt of the communications burden. A square cluster of four blocks has an additional benefit of lower power consumption if the comparison sequence shown in Figure 3.3 is done. The timing is such that only one input to the comparison block changes for each comparison cycle except for every fourth cycle when new inputs arrive. Thus, the power consumption is only four times larger than the parallel SIMD architecture.

Comparing all of the architectures presented above and determining which should be used for a given implementation can become a tremendous optimization problem. But some criteria are more important than others to achieve a "near optimum" design. For digital designs, the optimization criteria can be reduced to two measures, *area* and *performance* [19]. Area is essentially the size of the design measured in some convenient

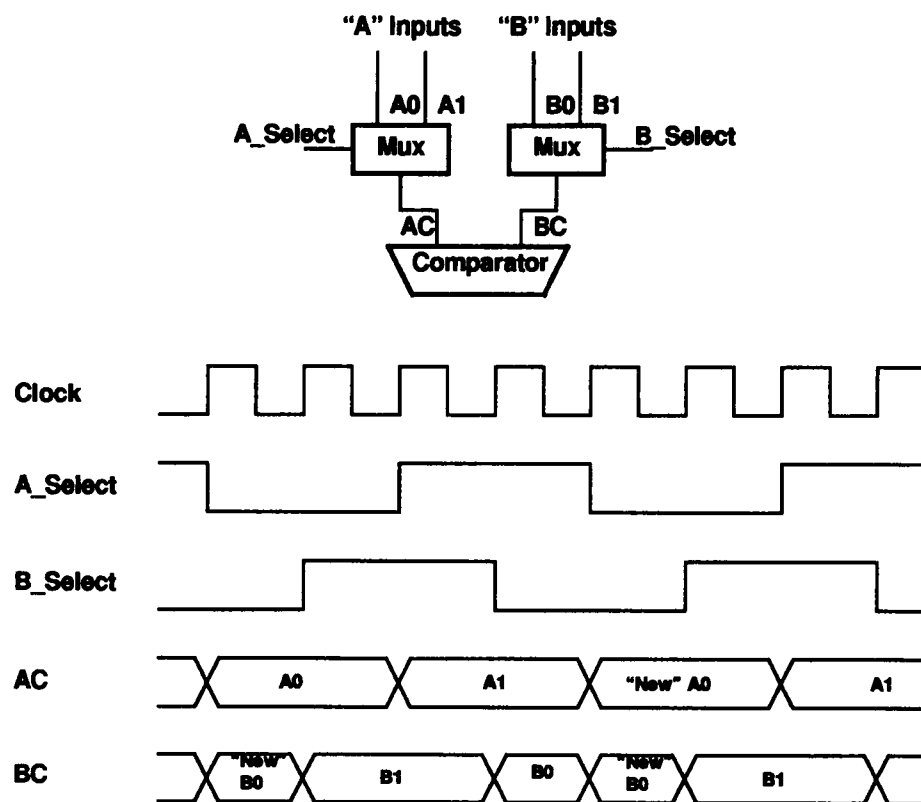


Figure 3.3. Comparison sequence for a four block cluster.

quantity. Performance is more evasive. It tends to be determined by the design *type* (i.e. combinatorial, synchronous, pipelined, etc.) as well as the personal bias of the designer. For combinatorial designs, performance tends to be measured in terms of propagation delay, the less the delay, the "better" the design. For synchronous designs, performance is measured in terms of cycle time, the lower the cycle time, the "better" the design. For pipelined designs, performance is measured in terms of latency (time required to execute the operations) [30] or throughput (rate of data production and consumption).

For this particular problem implemented as an ASIC in CMOS, the criteria chosen for comparing the architectures are area and power. The input and output stages are essentially the same for all of the architectures and thus would not influence a decision based on area or power. Since the major functional differences between the architectures is how and when the values are compared, the area of the architectures will be based on the number of comparison blocks used.

The power is a bit more nebulous. Using the same logic as for the area, a reasonable measure of architectural differences is the number of comparison cycles required per coincidence detection cycle. To understand this, consider that all of the architectures perform the same number of comparisons. But the *rate* at which the comparisons are performed varies depending upon the architecture. This rate is essentially a measure of how fast the comparison block must be (i.e. propagation delay). For CMOS, the power of a particular function is directly related to the rate at which it is operating. By using power as the measure instead of the rate, architectural differences which are related to the comparison blocks, such as data movement (shifting, etc.), are included as an increase or decrease in power consumption. An excellent example is the SIMD architecture with the serial implementation of the comparison block. The rate of the

Table 3.1. Coincidence detection architectures.

Architecture	Identification
SISD	A
SIMD with parallel comparators	B
SIMD with serial comparators	C
"Shift and compare"	C
"Cluster" with 4 comparisons	E

comparisons is the same as the parallel implementation, one comparison cycle in a coincidence detection cycle. But the power consumed for a comparison cycle is much more, namely 16 times that of the parallel implementation. Thus, the power consumed by the comparison blocks during a coincidence detection cycle is a better measure of the architectures.

Table 3.1 lists the architectures and assigns each a letter. Table 3.2 contains the criteria values estimated for each architecture where n is the number of comparisons required by the coincidence detection, m is the number of comparisons in a "row", and c is the number of comparisons in a cluster (second hybrid architecture). The limiting constraint for each architecture is also provided. This table provides a basis for comparing the architectures when the trade-off between area and power are the same, i.e. the designer feels that both are equally important. But a clearer understanding of the trade-offs between the architectures is possible with a plot where the trade-off between area and power are not the same.

Table 3.2. Coincidence detection architectural criteria values.

Architecture	Criteria Values		Limiting Constraint
	Area	Power	
A	1	n	Comparison cycle time
B	n	1	Comparison block size
C	$n/7$	16	Power
D	m	$2m$	Comparison cycle time
E	n/c	c	Comparison block size/cycle time

If weights are given to the criteria based on their importance to the designer, a measure of an architecture's performance, which will be called the *merit sum*, at a particular trade-off point is given by

$$w_{a,i} + w_{p,i} = 1.0 \quad (3.1)$$

$$A_{j,i} = w_{a,i}C_{a,j} + w_{p,i}C_{p,j} \quad (3.2)$$

where $w_{a,i}$ and $w_{p,i}$ are area and power weights, respectively, for the trade-off point i , $A_{j,i}$ is the merit sum of architecture j at the trade-off point i , and $C_{a,j}$ and $C_{p,j}$ are the area and power criteria values, respectively, for architecture j . The lower the merit sum, the "better" the architecture is at that trade-off point. Table 3.3 shows the criteria values for $n = 49$, $m = 7$, and $c = 4$. Note that the criteria values are normalized to the minimum area and power values, respectively. By normalizing the values to the minimums, the merit sums for each architecture are scaled relative to each other. Thus an architecture which requires twice the area as the smallest architecture will have a merit sum of 2.0 when area is most critical.

Table 3.3. Criteria values for $n = 49$, $m = 7$, and $c = 4$.

Architecture	Criteria Values	
	Area	Power
A	1	49
B	49	1
C	7	16
D	7	14
E	12.25	4

By plotting the merit sums versus $\log(w_{a,i}/w_{p,i})$ for various trade-off points, a better comparison of the architectures is possible. Figure 3.4 is a plot of the merit sums for each architecture over a range of weight ratios from 0.001 to 1000.0. At a ratio of 1.0, the area and power are deemed equally important. As one moves to the left to a ratio of 0.001, the power is deemed more important than the area. As one moves to the right to a ratio of 1000.0, the area is deemed more important than the power. Note, as one would expect, that the curves asymptotically approach the respective criteria values when that criteria dominates the decision.

From the plot, it is clear that architecture *A*, the SISD architecture, is the clear winner when size is crucial. Whereas, architecture *B*, the parallel SIMD architecture, is the clear winner when power (i.e. propagation delay) is crucial. But these are the extremes. In essence, the architectures which have more parallel features perform better when power/speed is most important. Whereas, the architectures which have more serial features perform better when area is most important. However, the architectures which

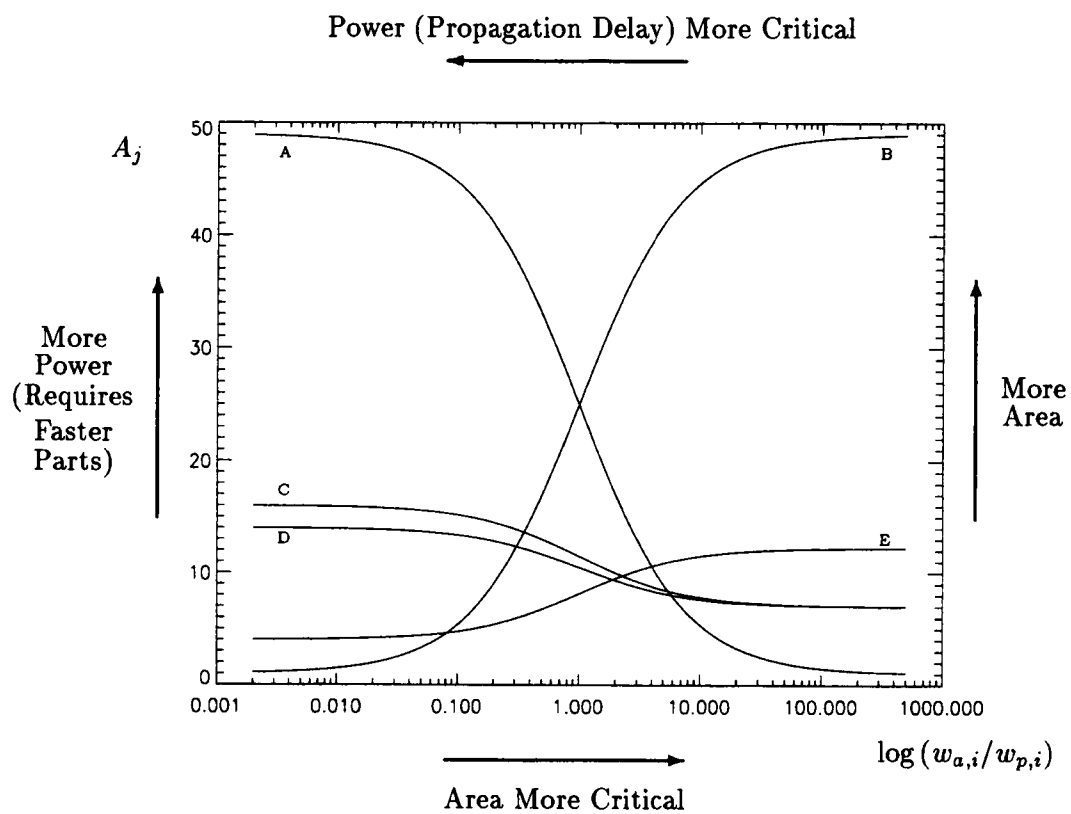


Figure 3.4. Plot of merit sums versus log of area/power weight ratio.

have flatter plots represent an inherently more equal trade-off between area and power within the architecture. In many cases, the trade-off between area and power is not as large as plotted (1000 to 1) but tends to be small (1 to <10 or <10 to 1) and not well defined. For these, the architectures with flatter plots and lower means are the better overall performers.

3.3 Design and Implementation of First Generation ASIC

The first generation coincidence detection ASIC was designed and implemented in 1988-1989 using a 2.0 micron CMOS gate array. The goal was 11 "A" and 11 "B" inputs giving a total of 56 module pairs. At the time of the design, only two architectures for the coincidence detection were considered: architecture *A*, the SISD, and *B*, the SIMD. Due to the speed required for the comparison block in the SISD architecture, the architecture chosen was Figure 3.2b with a four stage pipeline. The SIMD architecture is also well suited to handle the multiple concept. By having nearest neighbor communication of coincidences, the interconnect burden is reduced. However, the four stage pipeline was necessary due to worst case timing of the communication signals from one end of the array to the other. Figure 3.5 shows the four stage pipeline.

As shown by the architectural analysis above, the SIMD architecture exacts an area penalty on the design. To counter this penalty, an intense effort to reduce the size of the comparison block was performed. A serial comparison was considered but rejected due to power considerations. A parallel technique based on the fact that the values needed to be within ± 1 yielded a reasonable comparison block size. Note that the architectural analysis technique presented above had not been fully developed at the time of the

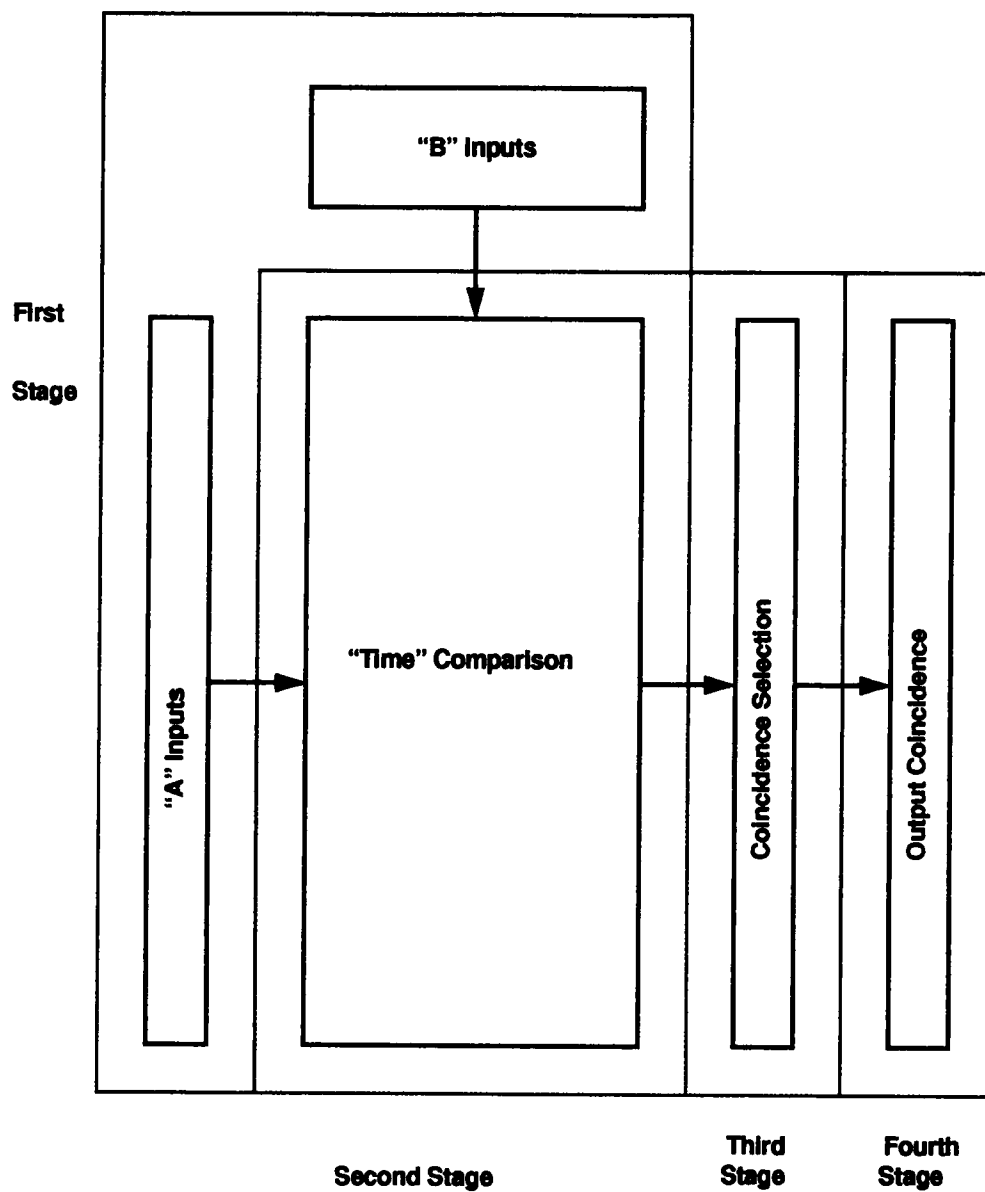


Figure 3.5. First generation coincidence detection ASIC pipeline.

design. If it had, more attention would have been placed on the serial comparison. Without this technique, the approach was more basic. A design would be produced and an estimate of gate count and speed would be found, usually on paper. Trade-offs were then considered and either the design pursued or another approach was tested.

The design for 11 "A" and 11 "B" inputs required approximately 27,000 gate equivalents. The cost, in 1989 dollars, was \$93,000 for NRE and \$330 per chip. This was prohibitive. But, due to the architecture chosen, the array can easily be broken into smaller pieces which are identical (an advantage of a systolic array). The design was reduced to a block of 4 "A" and 4 "B" inputs with the addition of signals to allow communication between chips. This design required approximately 10,000 gate equivalents and cost, in 1989 dollars, \$51,000 for NRE and \$30 per chip. Six chips are required to be equivalent to the original goal of 11 "A" and 11 "B" inputs. This brings the system cost to \$180 which is 54.5% of the original cost. The NRE is also 54.5% of the original as well. A scanner requires three sets of these which brings the total cost to \$540. The system which this ASIC replaced was built using commercially-available PAL and PROM devices and required three printed circuit boards with a total size of 480 sq. in. The system was connected to the scanner gantry with a set of high-speed, custom cable which was 75 feet long and cost approximately \$5000. With the introduction of the ASIC, the total circuitry required only 64 sq. in. and was mounted in the gantry. This eliminated all but 15 feet of the cable. The total cost savings, including amortization of the NRE, was about \$12,000 per system.

3.4 Design and Implementation of Second Generation ASIC

The second generation coincidence detection ASIC was designed and implemented in 1992-1993 using a 1.2 micron CMOS gate array. The goal was 10 "A" and 10 "B" inputs giving a total of 49 module pairs. At the time of the design, all of the architectures for the coincidence detection presented above were considered. However, since the architectural analysis technique had not been fully developed a more heuristic approach was taken.

The SIMD architecture had a success history which the other architectures had to overcome, but its size requirements reduced its appeal. The "shift and compare" architecture, Figure 3.2c, reduced the area but at the cost of increased power and speed. The cluster architecture, Figure 3.2d, was developed to improve to area qualities of the SIMD architecture and the power/speed qualities of the "shift and compare" architecture. Different cluster arrangements were examined by estimating gate counts, comparison speed, and power consumption. A cluster of four was chosen due to the propagation delay of the comparison block. A four stage pipeline similar to the first generation ASIC, Figure 3.5, was used. This allowed maximum time for the comparisons.

Again, the major area usage was in the comparison block. The second generation ASIC required a variable range on the comparison which increased the size as well. An intense effort to reduce the size of the comparison block was performed. A parallel technique based on the fact that the carry bits of an addition between the range and the two values to be compared are the critical information in the comparison yielded a reasonable comparison block size and speed [70].

The design for 10 "A" and 10 "B" inputs required approximately 18,000 gate equivalents. The cost, in 1993 dollars, was \$13,500 for NRE and \$75 per chip. Therefore, one second generation ASIC is equivalent to six first generation ASICs with a cost savings of nearly 60% and improved system performance [70].

CHAPTER 4

Crystal Identification ASIC

4.1 Description of the Problem

As described in Chapter 2, identification of the crystal (detector) in which a gamma ray was detected and the time of the occurrence are of primary importance in a PET scanner. Figure 4.1 shows a schematic representation of the processing associated with the detector electronics.

When a gamma ray penetrates the BGO, there is a high probability of interaction. When this interaction occurs, light is given off. The total amount of light given off is proportional to the energy of the interaction. The photomultiplier tubes (PMTs) detect this light and generate an electrical signal proportional to the intensity of the light. If this signal is of sufficient magnitude, the signals are integrated for a preset integration time. The integrated signals from the four PMTs are summed to produce the energy of the event. A ratioing scheme is used to determine the X/Y position of the interaction. The X and Y positions are given by

$$X_{Pos} = \frac{A + B}{A + B + C + D} \quad (4.1)$$

$$Y_{Pos} = \frac{A + C}{A + B + C + D} \quad (4.2)$$

where A , B , C , and D are the integrated signals from the corresponding PMTs and

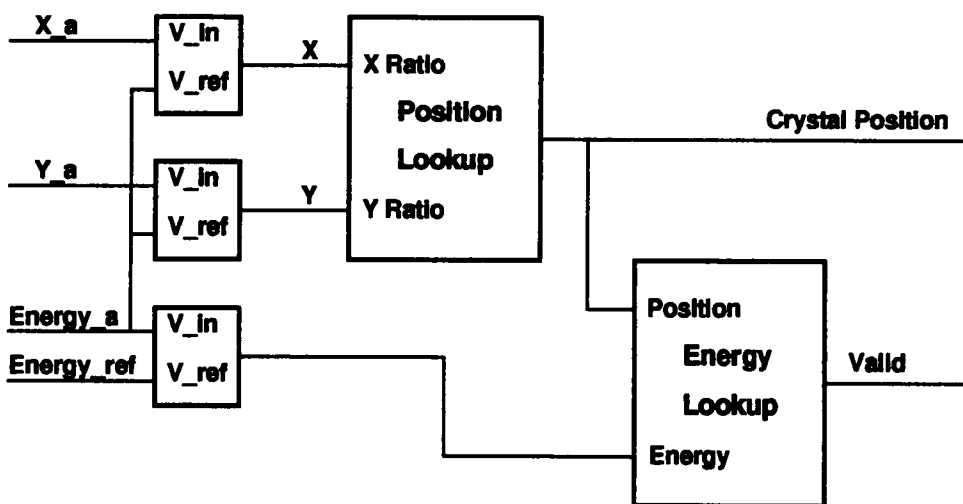
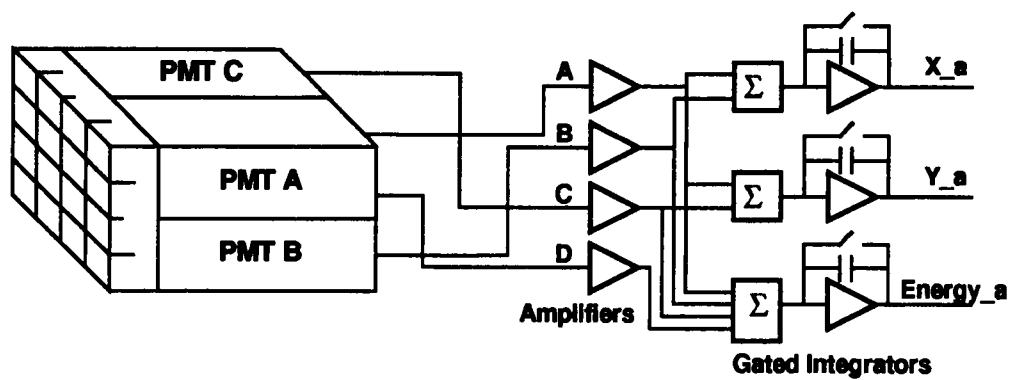


Figure 4.1. Schematic representation of PET detector processing.

X_{Pos} and Y_{Pos} are the X/Y position ratios. These ratios are determined during the analog-to-digital conversion by using the sum of all the PMTs as the reference voltage.

By placing a source in the field-of-view which uniformly interacts with all crystals, a histogram of the X/Y position ratios can be collected. Boundaries for the crystals are the halfway points between the peaks in the histogram. Once these boundaries are found, they are used to determine the crystal in which an event occurred during a PET scan. A typical histogram with boundaries is shown in Figure 4.2. A microprocessor controls the crystal identification system, defines the boundaries, and downloads them. Thus, a flexible interface and reprogrammability is required.

To be a valid event, an event must also be within a specific energy range. The energy ranges for each crystal are determined by taking an energy histogram for each crystal. An energy histogram is similar to the position histogram taken above except that the histogram of the energies for a specific crystal is collected. From the histogram, the peak of the gamma ray interaction can be found and cut-off points for the energy can be determined.

Using the boundaries and energy ranges of the crystals, an event can be tested to determine the crystal in which the interaction occurred and whether the energy is in the correct range. If the event is valid, the crystal and time of the event are output.

For the system described here, some specific requirements must be met. The events are obviously asynchronous and non-periodic. However, the output of a valid event must be synchronized with a master 256ns cycle. As with the coincidence detection ASIC in Chapter 3, there is no specific requirement that the event be available on the next cycle after it has occurred. But it is required that all events that occurred within a

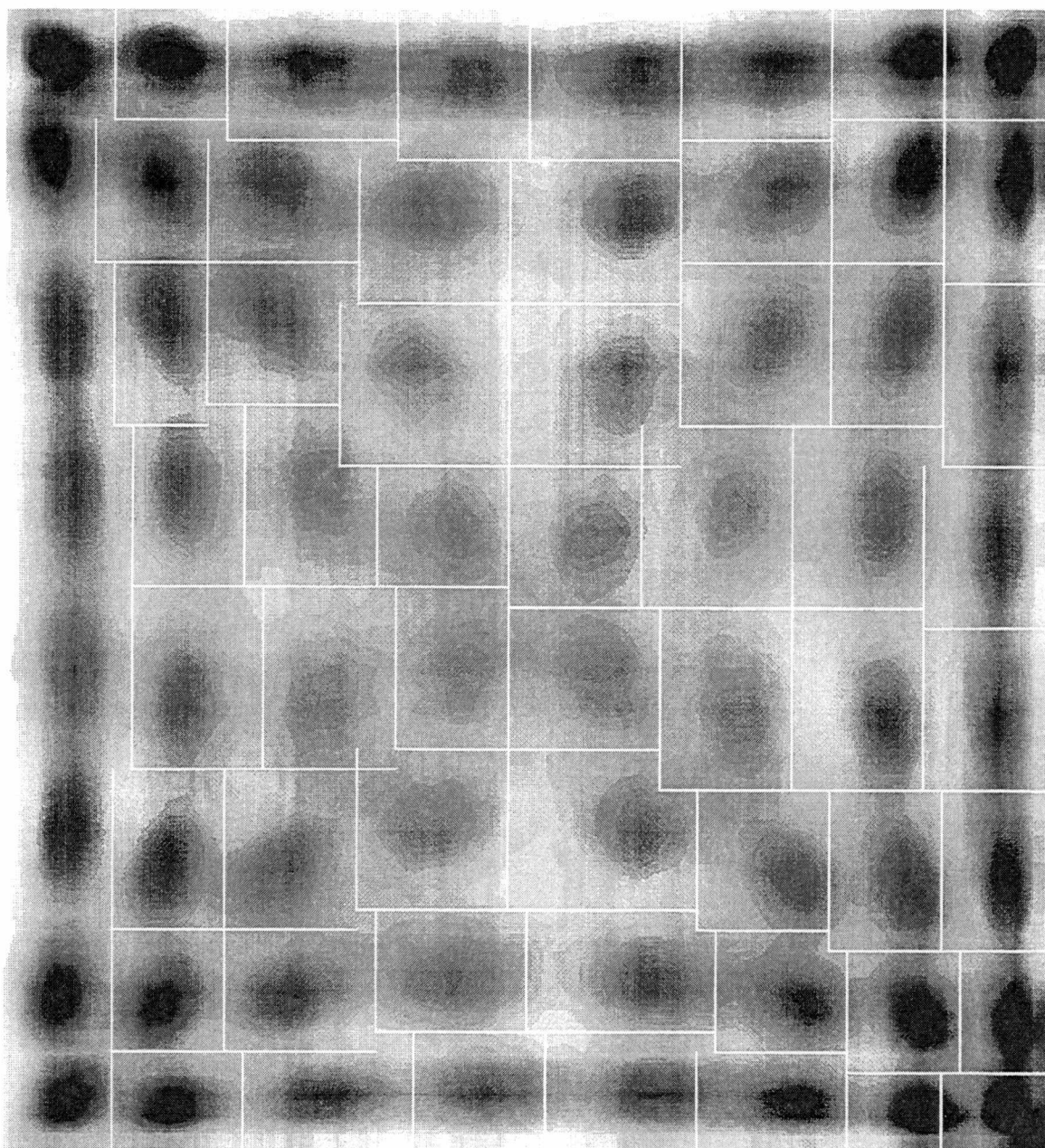


Figure 4.2. Position histogram with crystal boundaries.

given 256ns cycle arrive at the coincidence detection ASIC during the same 256ns cycle. Thus the crystal identification subsystem has a cycle time of 256ns.

The crystal identification subsystem is also responsible for generating the control signals for sequencing the analog subsystem when an event has occurred. These signals initiate, terminate, and reset integration of the signals from the PMTs. The analog-to-digital converters are also controlled. The X, Y, energy, and time presented to the crystal identification subsystem by the analog subsystem are 6-bit digit values.

Typically, more than one block of crystals share a common output channel. This requires a selection process when more than one block has an event during the same 256ns cycle. The process should be as unbiased as possible. Also due to the large number of blocks on a typical scanner (288), power consumption should be reduced as much as possible.

Some additional requirements which had very minor system impact have not been presented here for convenience. Further details can be found in [53, 8, 57]. The detector electronics system presented in [53] is an excellent description of the detector electronics required for PET. The crystal identification ASIC was designed to replace a large portion of the digital electronics in this system.

4.2 Analysis of Possible Architectures

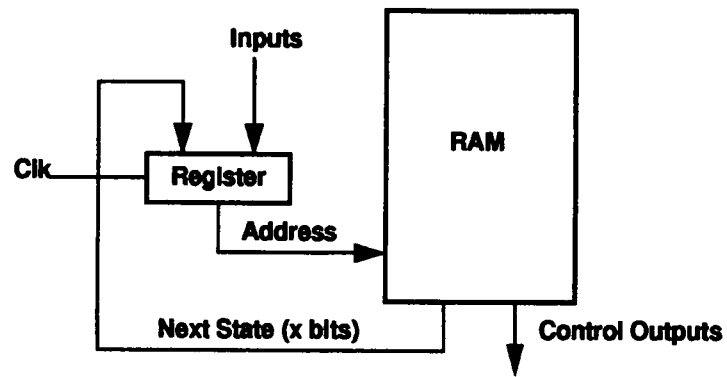
Using the above problem description, possible architectures may be analyzed. As with the coincidence detection ASIC, there is no input/output timing relationship required (relatively speaking). Therefore, pipelining may be utilized. From the description, a three stage pipeline is the minimum (input, crystal identification and validation,

output) and about 10,000 is the maximum. Taking the minimum three stage pipeline identified, each stage of processing can be considered separately. If additional stages of pipelining are found to be needed, it is easily added. Also, the sequencer for the analog subsystem is completely autonomous and can be treated separately from the crystal identification and validation section.

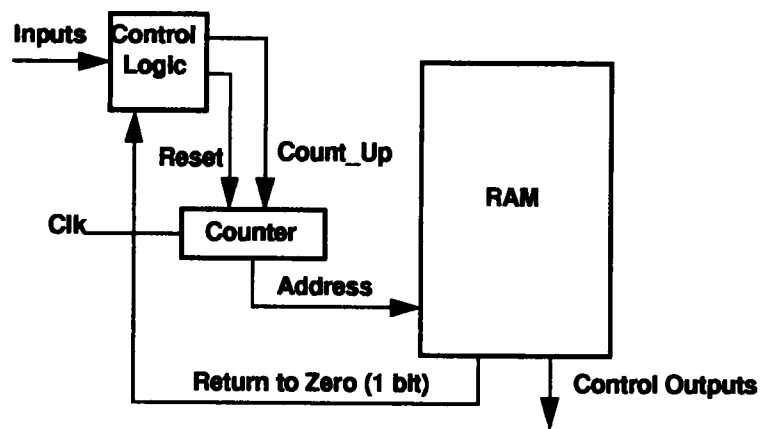
The sequencer for the analog subsystem is essentially a state machine. Many techniques for generating state machines in ASICs are available and will not be covered here. However, a major requirement of this sequencer was reprogrammability. This reduces the state machine options considerably. A state machine consisting of a random-access memory (RAM) was suggested as shown in Figure 4.3a. The problem this presents is the extra memory required for the next state address. An alternate solution which reduces this requirement to a single bit per state is shown in Figure 4.3b. This solution is based on the sequencing of the analog subsystem being sequential by nature with the length of the sequence and the outputs being variable. Thus, the next state address is redundant information and a single bit can be used to identify when the sequencer should return to the zero or "wait" state.

Similar to the coincidence detection ASIC, the input and output stages are very straightforward. Both may possibly perform two functions: buffering and format conversion. One additional function of the output is communication and selection of a single output between multiple crystal identification ASICs.

The main stage to be analyzed is crystal identification and validation where the actual processing is taking place. Various approaches to crystal identification can be defined. Figure 4.4 shows these architectural approaches.

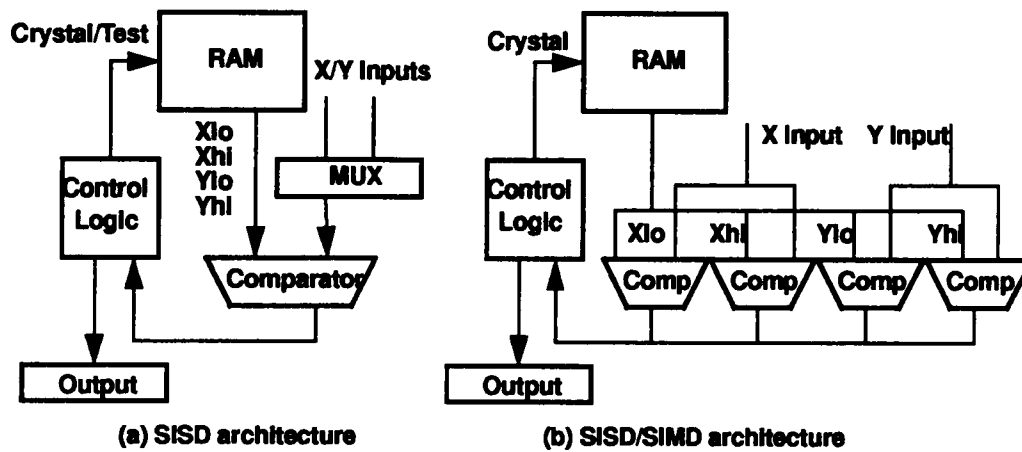


(a) Suggested sequencer state machine



(b) Alternate sequencer state machine

Figure 4.3. Sequencer architectures for the analog subsystem.



Captions:

CB = Comparison Block with Boundaries

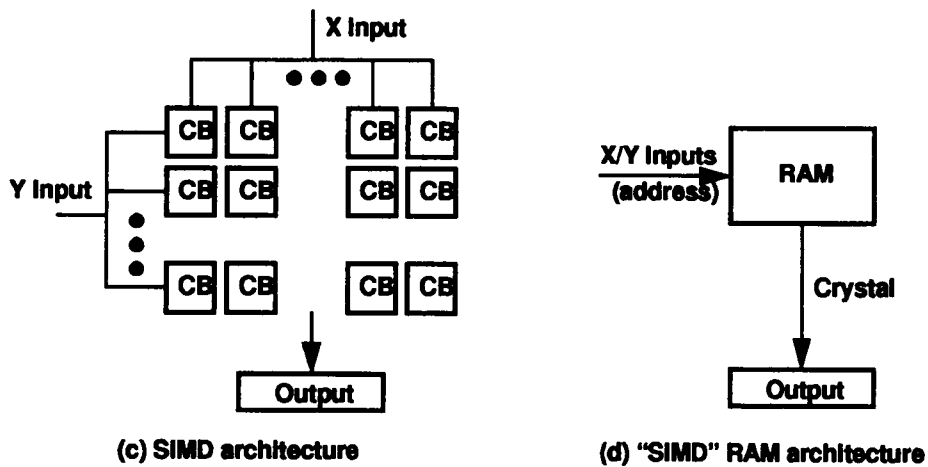


Figure 4.4. Various architectures for crystal identification.

Figure 4.4a shows a SISD approach to crystal identification which *loops* through the crystal positions and checks the X/Y position with the crystal boundaries. For this architecture, the time available for comparing the X/Y position with each crystal's boundaries is equal to the number of crystals to be checked divided by the cycle time. Assuming that the search is performed in a linear manner yields a worst case search of all the crystals (64 for the system described here). Therefore a "step-and-test" technique, similar to a successive approximation analog-to-digital converter, can be used to reduce the search. However, the boundaries are two dimensional forcing the stepping and testing of the X/Y boundaries to be performed separately, but possibly in parallel. The search begins at a "center" crystal. The X and Y input values are tested against the boundaries. Based on the result, either a step to the neighboring crystal in the appropriate X/Y direction is performed or the stepping is stopped. If a step is performed, testing of the current crystal boundaries is performed. Based on the result, either a step to the neighboring crystal in the appropriate X/Y direction is performed or the stepping is stopped. This step-and-test continues until the crystal is identified. This reduces the number of crystals to be checked to

$$T_{Crys} = \frac{\max(C_x, C_y)}{2} + 1 \quad (4.3)$$

where T_{Crys} is the number of crystals to be checked and C_x and C_y are the number of crystals in the X and Y directions, respectively. The addition of one is due to the boundaries being non-uniform. Figure 4.5 illustrates several searches with different X/Y inputs.

For a small number of crystals, this architecture has a size (spatial) advantage due to only one set of comparators being required. However to maintain the necessary throughput as the number of crystals increases, the available time to do the comparison will

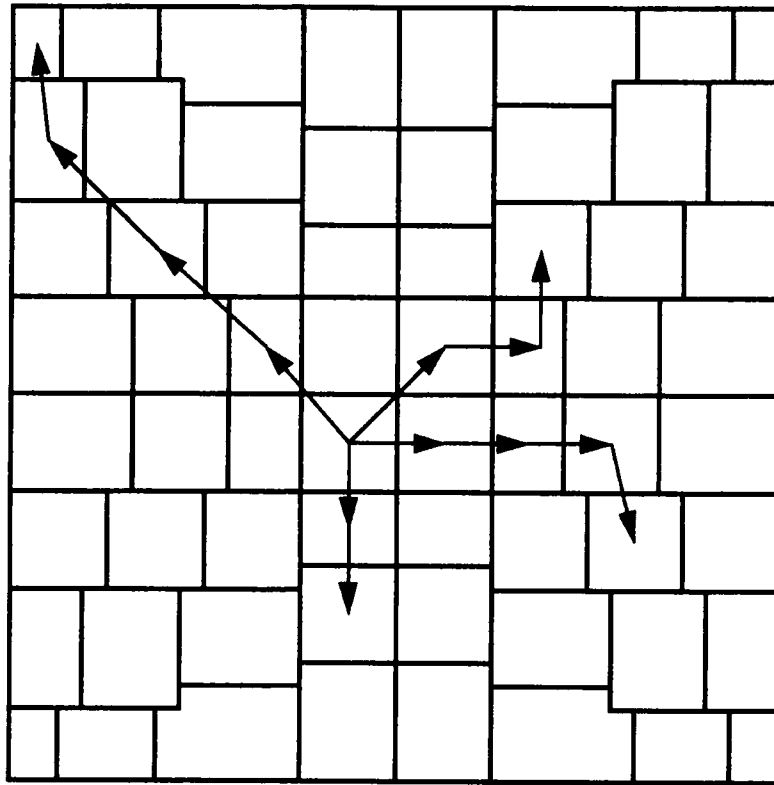


Figure 4.5. Step-and-test searching for crystal identification.

decrease. For architectural analysis, an estimate of the comparison time which includes switching and settling times can be used. Power consumption is also an issue with this architecture. As the number of crystals increases, the power consumption increases due to the comparison cycle time decreasing. Again references to power consumption are based on implementations in CMOS.

Figure 4.4b shows a hybrid SISD/SIMD architecture similar to the one above. The major difference is that the four comparisons at each crystal position are performed in parallel. This reduces the power consumption, but increases the size.

A SIMD type architecture is shown in Figure 4.4c. For this architecture, the boundary comparison has a full cycle time to complete, but there must be a comparison block for each crystal. Each comparison block has four comparators and a small amount of logic to define which crystal has the X/Y values within its boundaries. Thus it has a temporal advantage, but a spatial disadvantage. As with coincidence detection, it may be possible to use this temporal advantage to produce a size reduction. A serial comparison of the data may be performed instead of a parallel comparison (which has been assumed in these discussions). The trade-off becomes one of power consumption as in the SIMD architecture for coincidence detection. If there are eight bits of propagation data (two for each comparison), the above serial comparison block will consume about 48 times the power as the parallel comparison block, but it will be about 1/6 to 1/8 the size.

For the above architectures, the amount of memory used to store the boundaries for the crystals can be reduced by noting that adjacent crystals share a boundary. Thus the X_{High} boundary for one crystal is the X_{Low} boundary for the adjacent crystal in

the increasing X direction. Thus instead of requiring

$$M_{Crys} = 4 * C_x * C_y$$

where M_{Crys} is the amount of memory required in X/Y word widths and C_x and C_y are the number of crystals in the X and Y directions, respectively, only

$$\begin{aligned} M_{Bound} &= (C_y * (C_x + 1)) + (C_x * (C_y + 1)) \\ &= (2 * C_x * C_y) + C_x + C_y \end{aligned}$$

where M_{Bound} is the amount of memory required in X/Y word widths, are required.

Figure 4.4d shows another architecture which can be considered a SIMD architecture. In this architecture, a RAM is used to identify the crystal with no testing of the boundaries required. A crystal value is stored for every possible value of X and Y. The address of the RAM is the X/Y input values. Thus, the crystal is identified by selecting a single RAM location. As with the other SIMD architecture, this architecture has a temporal advantage, but a spatial disadvantage. For each bit increase in the width of X and Y, the amount of memory required increases by a factor of four. This is the approach taken in the implementation by Moyers [53].

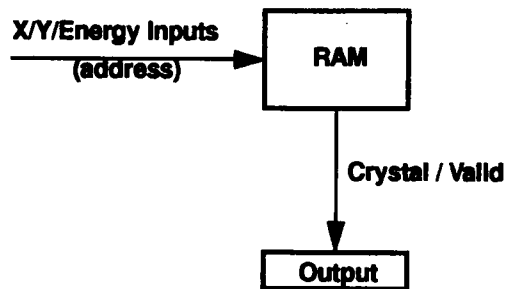
Validation of the event for the selected crystal can be performed either during or after the crystal identification. Regardless of when it is done, the memory requirement is the same, essentially three energy values for each crystal, except for the RAM-based architecture. The architectural alternatives are parallel or serial comparators and a single comparator or comparators for each energy.

In the RAM architecture if the validation is done during the crystal identification stage, the depth of the RAM will increase by 2^{width} where *width* is the number of bits for the energy and the width of the RAM will only increase by a single bit per location

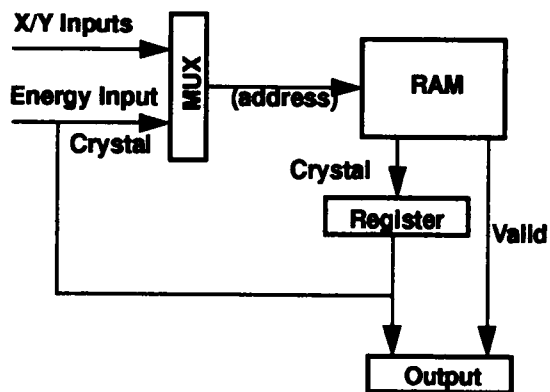
to indicate a valid event or not. Whereas, if the validation is done after the crystal identification, the same RAM used for crystal identification can be used for validation. An additional bit per location is required and the address inputs are changed from X and Y to the crystal location and the energy [53]. The second approach requires a RAM which can be accessed in half the time, but is considerably smaller. Figure 4.6 shows the two different architectures.

As with coincidence detection, the optimization criteria for the crystal identification and validation architectures are chosen to be area and power. The input, output, and sequencer sections are essentially the same for all of the architectures and thus would not influence a decision based on area or power. Since the major functional differences between the architectures is how and when the values are compared, the area of the architectures can be based on the number of comparison blocks used. However, the amount of memory used for storing either the crystal boundaries or the crystal locations varies between the architectures. Thus the area will be based on the total memory and number of comparators used by the architectures. Since a six-bit comparator is much larger than a six-bit memory location, an "equalization" factor needs to be used. For the ASIC described below, a six-bit comparator is approximately two times as large as a six-bit memory location. Thus, each six- to eight-bit memory location will be given an area value of one and each six- to eight-bit comparator will be given an area value of two.

Using the same logic as was used for coincidence detection, a reasonable measure of the power for crystal identification and validation is the rate at which memory accesses and comparisons are performed during a crystal identification and validation cycle. For



(a) "SIMD" RAM architecture with energy validation



(b) "SIMD" RAM architecture with pipelined energy validation

Figure 4.6. Non-comparator, RAM-based energy validation architectures.

Table 4.1. Crystal identification architectures.

Architecture	Identification
SISD (one parallel comparator)	A
SISD/SIMD with parallel comparators for X/Y/Energy	B
SISD/SIMD with parallel comparators, pipelined Energy	C
SIMD with parallel comparators (X/Y/Energy)	D
SIMD with parallel comparators (X/Y), pipelined Energy	E
SIMD with serial comparators	F
"SIMD" RAM with X/Y/Energy as address	G
"SIMD" RAM with X/Y with pipelined Energy	H

the architectural analysis used here, the power used to access a memory location can be assumed to be equal to that used to compare two values.

Table 4.1 lists the architectures and assigns each a letter. Table 4.2 contains the criteria values estimated for each architecture where N_{Crys} is the number of crystals in the block ($C_x * C_y$ which are the number of crystals in the X and Y directions, respectively), T_{Crys} is from Equation 4.3, M_{Bound} is Equation 4.4, and *width* is the bit width of the X, Y, and energy values. The limiting constraint for each architecture is also provided. This table provides a basis for comparing the architectures when the trade-off between area and power are the same, i.e. the designer feels that both are equally important. But a clearer understanding of the trade-offs between the architectures is possible with a plot where the trade-off between area and power are not the same.

Table 4.2. Crystal identification architectural criteria values.

Architecture	Criteria Values			Limiting Constraint
	Comparators	Area Memory	Power	
A	1	$M_{Bound} + 3N_{Crys}$	$4T_{Crys} + 3.0$	Comparison cycle time
B	7	$7N_{Crys}$	T_{Crys}	Comparison block size/cycle time
C	7	$M_{Bound} + 3N_{Crys}$	T_{Crys}	Comparison cycle time
D	$7N_{Crys}$	$M_{Bound} + 3N_{Crys}$	1	Comparison block size
E	$4N_{Crys} + 3$	$M_{Bound} + 3N_{Crys}$	1	Comparison block size
F	$\frac{N_{Crys}}{7} + 3$	$M_{Bound} + 3N_{Crys}$	48	Power
G	0	2^{3width}	1	RAM size
H	0	2^{2width}	2	RAM size and speed

Table 4.3. Criteria values for 64 crystals and six-bit input values.

Architecture	Criteria Values	
	Area	Power
A	338	23
B	462	5
C	350	5
D	1232	1
E	854	1
F	360.3	48
G	262144	1
H	4096	2

Table 4.3 shows the criteria values for 64 crystals (8x8 block) and six-bit X, Y, and energy values. Note that the criteria values are normalized to the minimum area and power values, respectively. By normalizing the values to the minimums, the merit sums for each architecture are scaled relative to each other. Thus an architecture which requires twice the area as the smallest architecture will have a weighted sum of 2.0 when area is most critical.

Using Equations 3.1 and 3.2, a measure of an architecture's performance can be found at a particular trade-off point. By plotting the merit sums versus $\log(w_{a,i}/w_{p,i})$ for various trade-off points, a better comparison of the architectures is possible. Figure 4.7 is a plot of the merit sums for each architecture over a range of weight ratios from 0.001 to 1000.0. At a ratio of 1.0, the area and power are deemed equally impor-

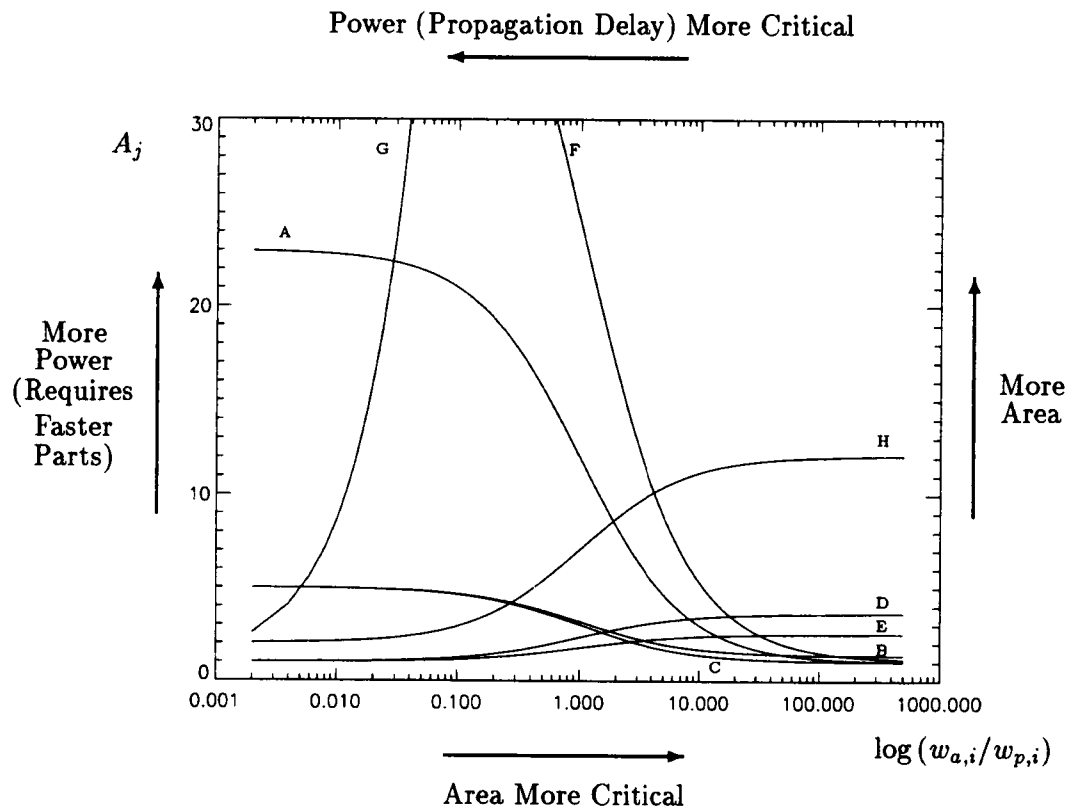


Figure 4.7. Plot of merit sums versus log of area/power weight ratio.

tant. As one moves to the left to a ratio of 0.001, the power is deemed more important than the area. As one moves to the right to a ratio of 1000.0, the area is deemed more important than the power. Note, as one would expect, that the curves asymptotically approach the respective criteria values when that criteria dominates the decision.

From the plot, it is clear that architecture *G*, single RAM with address inputs of *X*, *Y*, and energy, is not a reasonable architecture to consider when the other architectures are available (the merit sum when area is critical is literally off the plot). However, the other single RAM architecture [53], *H*, compares quite favorably with the other architectures, using only twice the power and about 10% more area than architecture *D*, the SIMD parallel comparator architecture. As with coincidence detection (and most designs), the architectures which have more parallel features perform better when power/speed is most important. Whereas, the architectures which have more serial features perform better when area is most important. However, the architectures which have flatter plots represent an inherently more equal trade-off between area and power within the architecture. This is represented here by architecture *C*, the serial search with parallel comparators, a hybrid SISD/SIMD, which has the flattest and lowest overall merit sums of any of the architectures.

4.3 Design and Implementation of ASIC

The crystal identification ASIC was designed and implemented in 1991-1992 using a 1.0 micron CMOS gate array in less than a nine month period. The goal was to implement as much of the existing digital detector electronics as possible for an 8x8

block. This included the sequencer, counters for event and deadtime measurements, arbitration logic, and crystal identification and validation.

As with coincidence detection, the above architectural analysis had not been fully developed at the time of this design. Therefore, a more heuristic approach was taken. A design would be produced and an estimate of gate count and speed would be found, usually on paper. Trade-offs were then considered and either the design pursued or another approach was tested.

From this heuristic approach, a serial search with parallel comparison, architecture *B*, was chosen. The SIMD architecture, architecture *E*, was considered along with various ways to reduce the area, but rejected because of the area required. The fully serial search architecture, architecture *A*, was also considered. It was rejected since the difference in area between it and *B* was so small and the power consumption so much greater. The heuristic approach yielded the same answer but required a much more detailed analysis of the designs.

The actual implementation of the crystal identification ASIC varied slightly from architecture *B*. The method of storing boundaries described in the architectural section above was not used. Instead, the boundaries for each crystal were stored separately to reduce the logic required to access a crystal's boundaries. This was chiefly done to reduce design effort due to time-to-market pressures. Figure 4.8 shows a block diagram of the overall crystal identification ASIC design.

The design for the crystal identification ASIC required approximately 27,000 gate equivalents implemented in a 37,000 gate, 1.0 micron CMOS gate array. Of the 27,000 gates, less than 8,000 were for logic and the remaining 19,000 were for the RAMs.

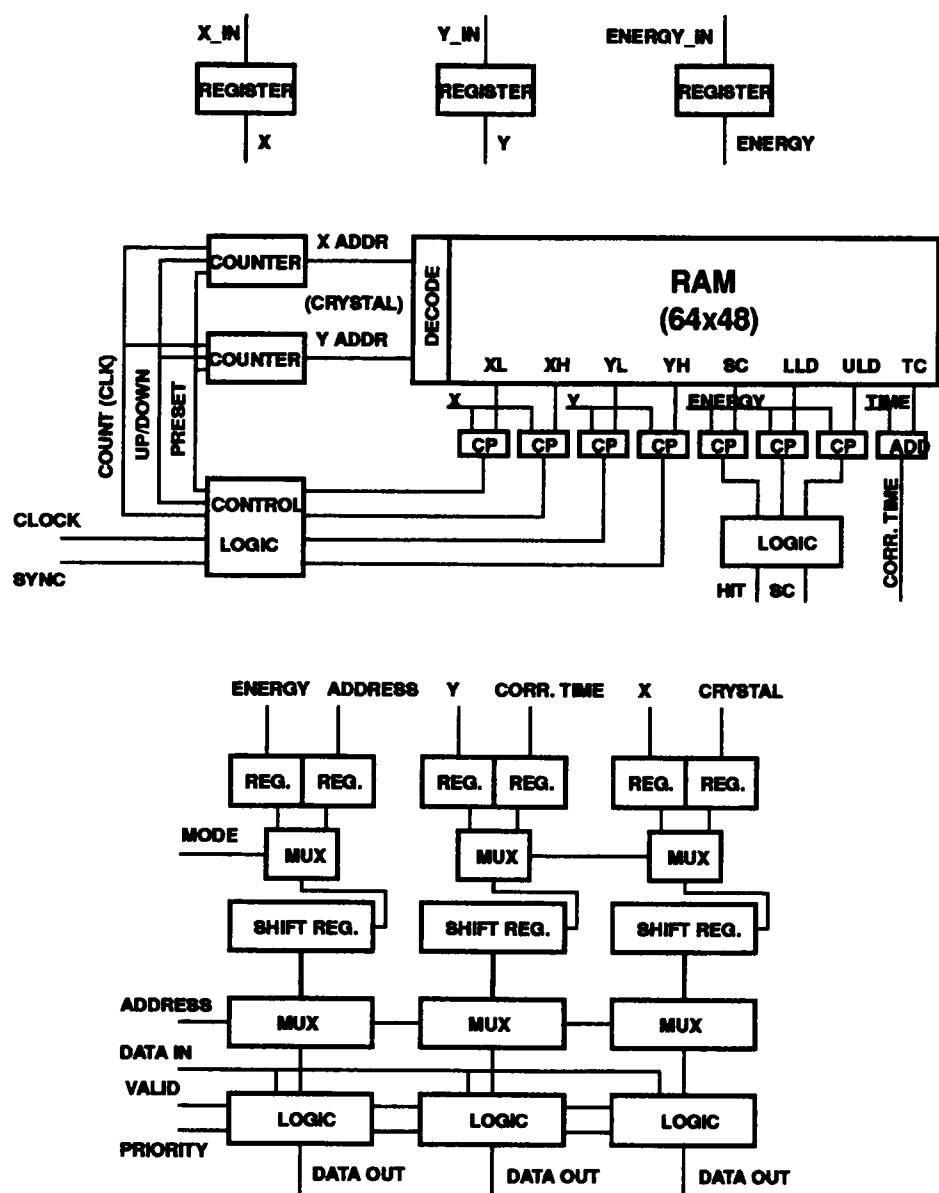


Figure 4.8. Block diagram of crystal identification ASIC.

Figure 4.9 shows the crystal identification ASIC floorplan. The cost, in 1992 dollars, was \$37,500 for NRE and \$14.65 per chip.

The ASIC is packaged in a 68-pin PLCC and consumes less than 0.5W with a system clock of 32MHz. Compared to the design described in Moyers [53], a system designed with this ASIC has twice the number of blocks, uses 60% less power, reduces the IC count by a factor of 4, reduces board space by a factor of 3, and at half the cost per channel. However, there were other benefits which are not as readily apparent.

In the 1991 time frame in order to maintain a market lead, a PET scanner with smaller size detector crystals but twice the number of detectors needed to be designed. If the existing electronics were used, larger power supplies, more cooling, and larger space would have been needed. This would have forced a major mechanical redesign. However by implementing the ASIC, the mechanical redesign was avoided. Since the power consumption is reduced, the power supply and cooling system did not have to be changed for twice the number of detectors.

Two other aspects of this design should be mentioned. Both involved the design methodology. The first involved a desire for "vendor-independence" of the ASIC design. Design reuse and "vendor-independence" for ASIC designs has received a great deal of attention in recent years. This is mainly due to two reasons: a desire to receive competitive pricing and the potential of needing a second source. Both of these reasons can be reduced to a single phrase: a desire to not be tied to a single vendor. This has lead many to see hardware description languages, i.e. VHDL, with retargeting of the design to different vendors as a solution. It is an excellent solution. However, HDL compilers, simulators, and retargeters are expensive and may not be available for some users. This was the case for this ASIC design, as well as the coincidence detection ASIC.

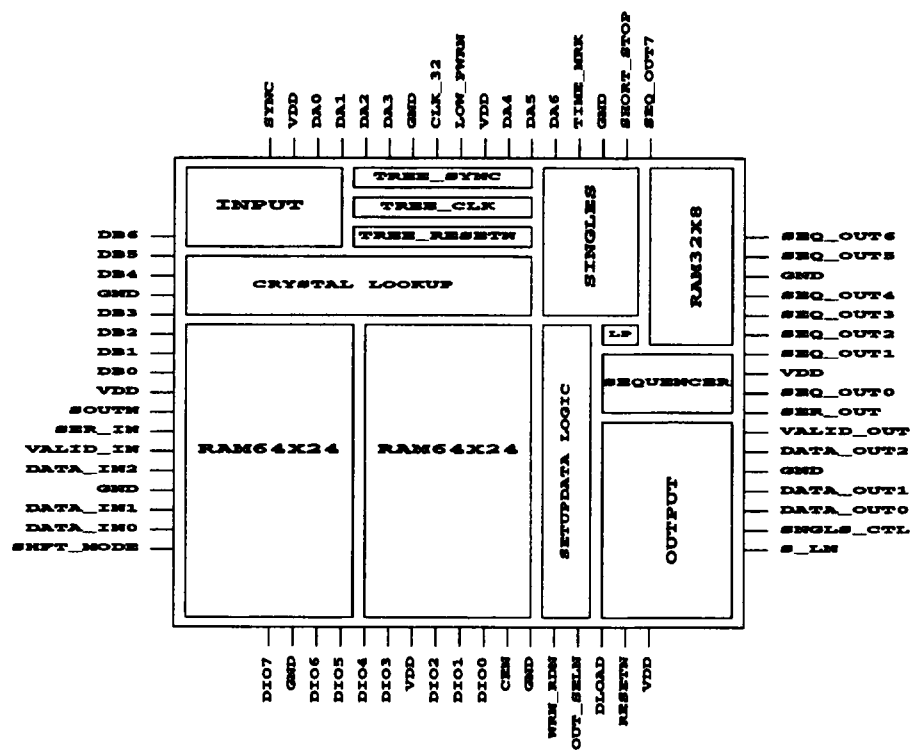


Figure 4.9. Crystal identification ASIC floorplan.

However with a proper approach, schematic capture and simulation packages can yield a "vendor-independent" solution with very little effort. Vendors supply, without charge, schematic symbols and simulation models for their gate arrays and standard cells in a "library" which is added to the search path of the schematic package. The typical design approach is to design everything using these cells exclusively. A more flexible approach is achieved if the designer follows a hierarchical approach and creates an intermediate library of cells which reference the vendor's cells. The designer then only designs using these intermediate cells. These intermediate cells correspond to the low level cells of the design, i.e. the ANDs, NANDs, NORs, flip-flops, comparators, etc., from which the rest of the design is constructed.

To change to a different vendor, the designer only has to modify the intermediate cells to use the new vendor's cells and change the search path. All aspects of the design can remain the same including the test vectors. With several different vendors' libraries available in this manner, a simple change to the search path allows a different vendor to be selected. When implementing these intermediate cells for different vendors, similar devices, i.e. similar loads and drives, should be used to minimize any loading and drive changes when a different vendor is used.

In the fall of 1994, the user of the crystal identification ASIC (CTI, Knoxville, TN) was notified by the vendor who produced the ASIC that they were shifting their market emphasis and the quantity being purchased was below their desired market level. Thus, they would end production of this part and were offering a "final" production run. A decision was made to seek a second vendor to provide future production. Having done the design in the "vendor-independent" manner described above, the entire process from after vendor selection to production parts required less than six calendar months. Of

this time, nearly three months were for production lead times since ten prototypes were received within three months. Of the three months, two weeks were required from final sign-off to arrival of the prototype parts. Of the remaining time, over a month was spent getting an up-to-date library of parts and simulation models for the RAM from the vendor. Conversion and testing required minimal effort by the designer. There were some slight additions to provide an extra time bit output but the architecture and major components remained the same. The design for the crystal identification ASIC from the second vendor was implemented in a 37,000 gate, 0.8 micron CMOS gate array in the spring of 1995. The cost, in 1995 dollars, was \$51,000 for NRE and \$9.50 per chip.

By having done the design in this manner, the impact to the user when a second vendor was required was minimal. The cost was the second NRE which might seem substantial, but there was no production time lost, not a long development process, complete compatibility, and a lower per chip cost.

The second aspect of the design methodology that should be mentioned involved the generation of test vectors for simulation and testing. Normally one writes input vectors using commands to the simulator. Some simulators offer "C"-like constructs to aid in looping and other tasks. However, this does not give a flexible means of generating test vectors, a CAD tool independent solution, or additional verification of the design. The approach taken with this design was to model the function of the ASIC using code written in "C".

By modeling the functionality in "C," input and output vectors were easily created for *any* simulator by recompiling the code. Routines were created which produced random inputs and outputs based on those inputs. One of the tests which was generated by the model and then simulated by the simulator package on the design used over *one*

million test vectors. Not only did the model provide test vectors, it helped verify the design by serving as a separate implementation. During simulation of vectors generated by the model, any discrepancies between what the simulator found and what it was told to expect are flagged as errors. This leads to the question of which design is at fault: the hardware design or the software model. In resolving these errors, a better understanding of the design and an improved design are achieved. To illustrate this, the vectors which were given to the vendor to test the ASICs during production were reported by the vendor to yield 99.91% toggle coverage of all internal nodes. Toggle coverage means that a node has made both a 0 to 1 and 1 to 0 transition during testing. This set of test vectors is typically quite small, usually < 64,000 vectors. Having the model available gave a better understanding of the design which led to better testing of the ASIC.

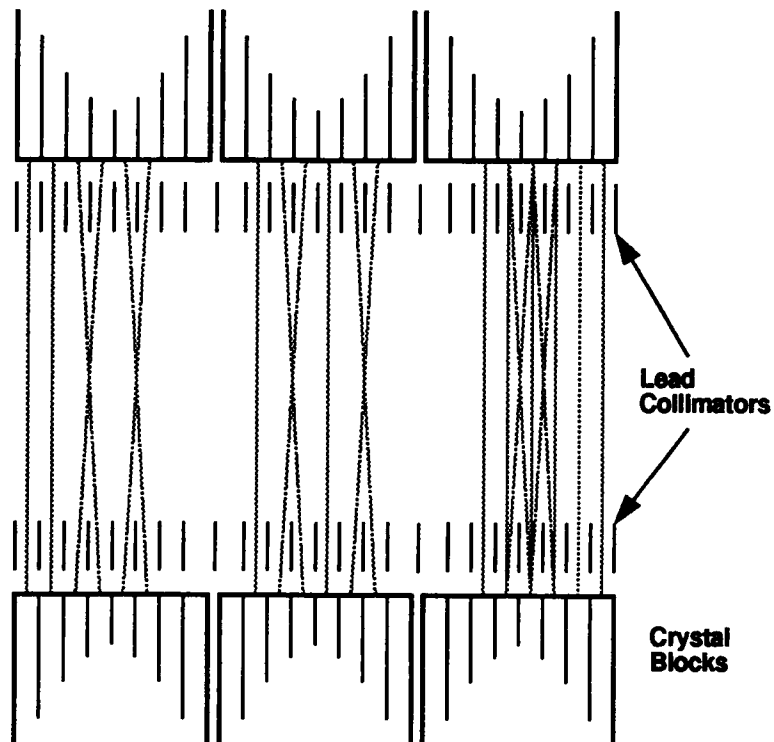
CHAPTER 5

3D Backprojection

5.1 Description of the Problem

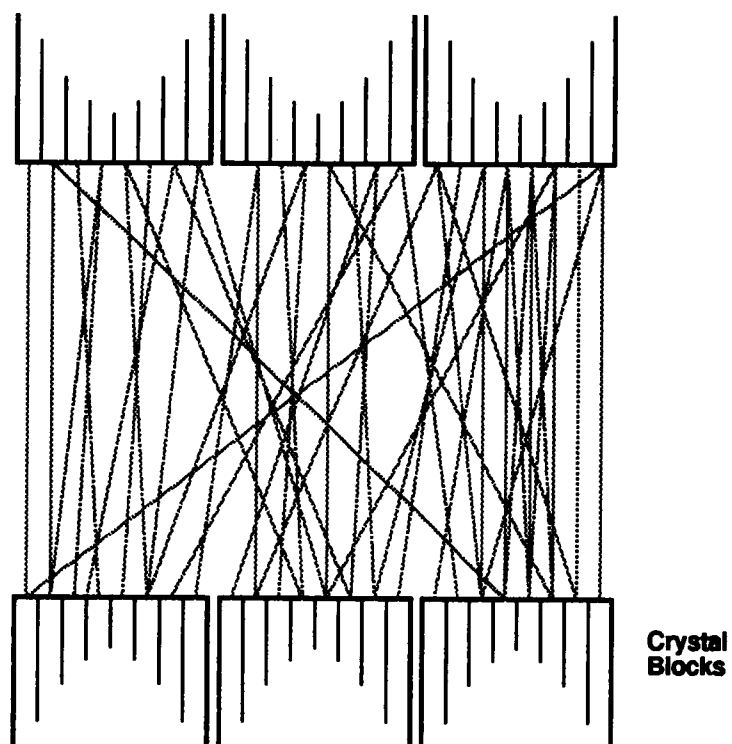
Image reconstruction is the last stage in the PET imaging process and can be the most time consuming. The reconstruction process described in the overview is known as 2D filtered backprojection. It assumes that there are axially distinct “planes” which when stacked produce a volume. Each plane has a separate sinogram and can be reconstructed separately. In an attempt to insure that the planes are distinct, the acquisition of the events is setup to only allow coincidence detection between single events within ± 1 plane of each other. Also, a lead collimator is placed in the field-of-view which restricts single events to these planes. Figure 5.1 illustrates this graphically.

3D PET, and thus 3D backprojection, removes these restrictions. The lead collimator is removed and coincidence detection between single events which are not in the same plane is permitted, Figure 5.2. The LORs generated from these coincidences have an axial angle component. The 2D pixel elements are now 3D volume elements called “voxels.” The voxel values are the total contributions (sum) from all LORs which pass through the voxel. Note that the 2D filtered backprojection is a special case of the 3D backprojection. The filtering for the 2D case is a 1D filter, however, the filtering for 3D is applied axially as well as radially [16]. This presents a problem.



Possible Coincidences (not all shown): _____

Figure 5.1. 2D coincidence detection configuration.



Possible Coincidences (not all shown): _____

Figure 5.2. 3D coincidence detection configuration.

As the axial angle becomes steeper, the number of “planes” in the view becomes smaller. The filter derived by Colsher [16] requires that the number of planes remain the same throughout the angular extent in the axial direction. Several approaches to circumvent this problem have been proposed, but the most widely accepted is the “reprojection” algorithm by Kinahan et al. [61, 39]. In the reprojection algorithm, the LORs of the 3D data set which correspond to the 2D set are used to form an *estimated* volume using 2D filtered backprojection. The projections that are not measured by the scanner are estimated by *forward-projecting* through the volume. Forward-projection is the inverse of backprojection, i.e. the projection value is the sum of the voxels the LOR passes through. With all planes present, the Colsher filter can be applied and 3D backprojection performed.

The 3D backprojection process proceeds as follows. For each axial angle, each transaxial angle has a 2D view of projections which are corrected and filtered. Similar to the 2D approach, a line is projected from each voxel to the 2D view. The value at the point of intersection in the view is added to the voxel. This is performed for each voxel at each transaxial angle for each axial angle. To give some idea of the magnitude of the backprojection process for 3D, consider a commercial PET scanner, the EXACT HR (CTI, Knoxville, TN) in which a typical reconstructed volume has $47 \times 128 \times 128$ voxels. This scanner’s default 3D mode of acquisition has 196 transaxial angles and 5 axial angles. Thus, the backprojection operation is performed

$$5 * 196 * 47 * 128 * 128 = 754,647,040 \text{ times}$$

This is just the backprojection operation. The forward projection operation is about 30% of this number, but requires more steps to be performed. In the implementation

of 3D backprojection presented here, the forward projection accounts for the largest amount of computation time (45%) followed by the backprojection (40%).

5.2 Analysis of Possible Architectures

Using the above algorithm description, possible algorithm architectures may be analyzed. However in this case, the design is for a *software* implementation on a multiprocessor system and, as Hwang [34] points out, "all machine-implementable algorithms must be architecture-dependent." This means that the implementation of the algorithm is dependent on the architecture of the multiprocessor system on which it will be executed. Thus, the architecture of the multiprocessor system must be described.

Figure 5.3 shows a schematic representation of the multiprocessor system on which the 3D backprojection is to be executed. The Sparc processor is a general purpose CPU running SUN-OS (a variant of UNIX) and the i860's are essentially "co-processors" available via the VME bus. The Sparc is responsible for initiating, synchronizing, and terminating the processing on the i860s. The memory for each i860 is mapped into the memory space of the Sparc and is available to both the i860 and the Sparc but not directly to the other i860s. There is a direct-memory access (DMA) transfer available on the i860's which, if properly configured, can be used to move data from one i860 to another.

Communication and I/O between the Sparc and each i860 is via message passing or the shared memory with locking mechanisms available. The message passing is restricted to single 32-bit values, thus it is used mainly for communication and synchronization. There is no direct communication ability between i860's. However, the vendor of the

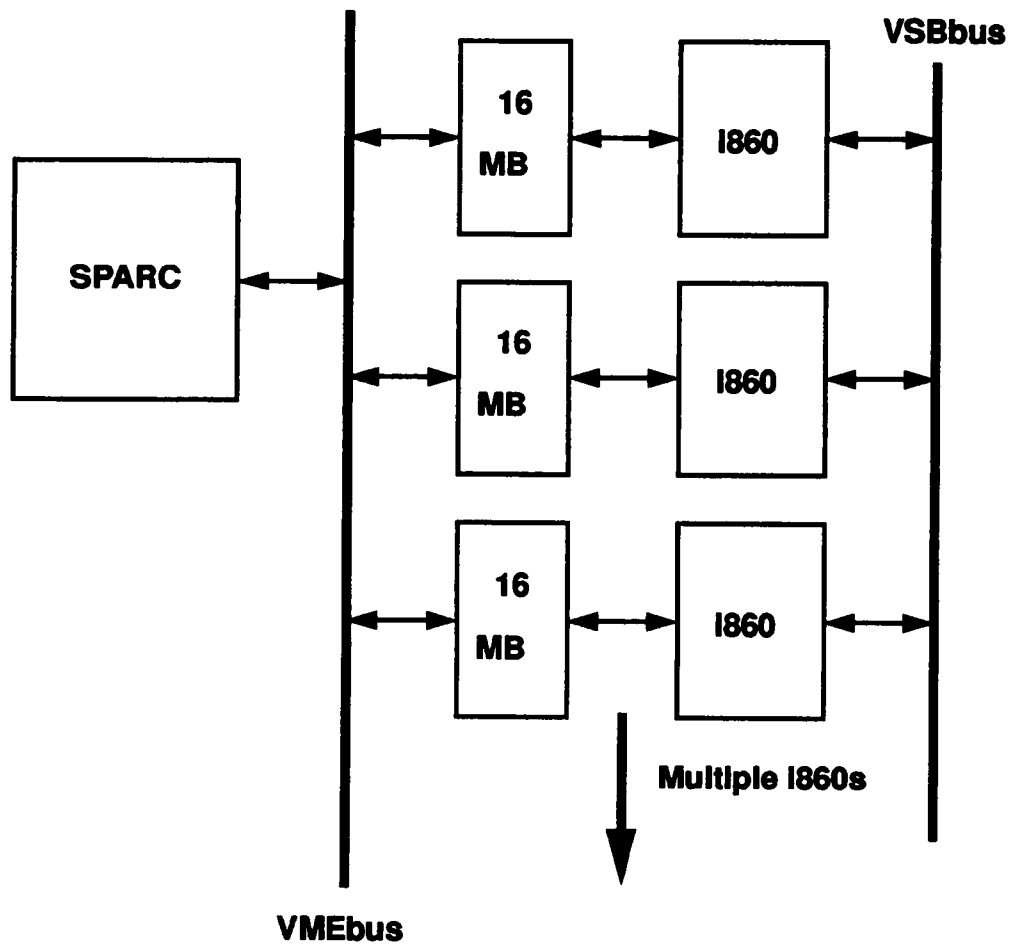
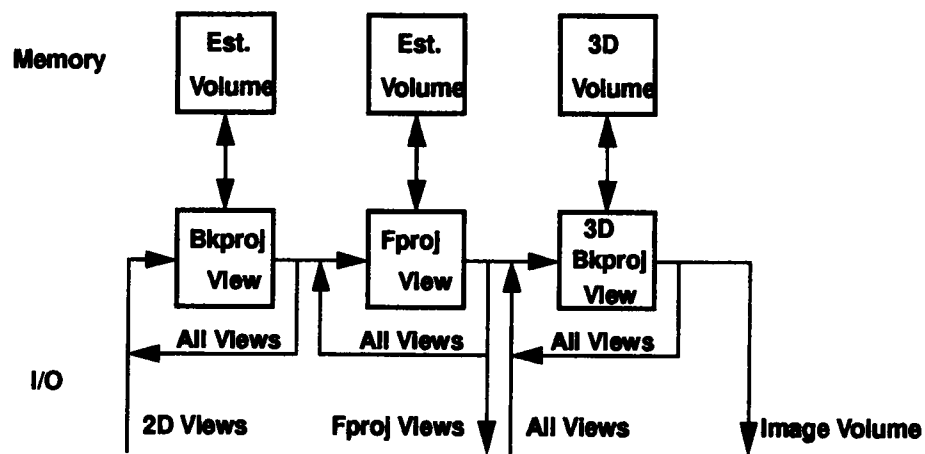


Figure 5.3. Multiprocessor system for 3D backprojection.

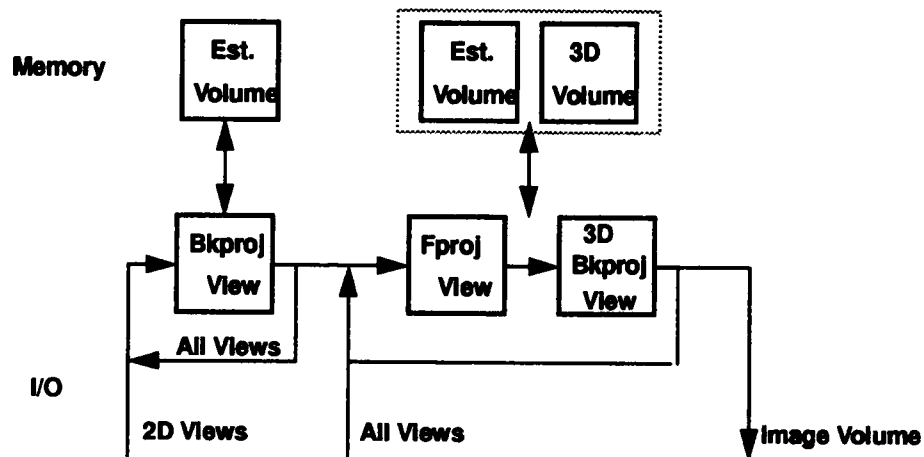
i860 processor cards indicates that a future release of the control software will have both communication and shared memory access between i860's.

A distributed memory model is more appropriate for this multiprocessor system than a shared memory model. However, the shared memory between the i860's and the Sparc provides more flexibility thus making it more of a hybrid memory model. Through the design and implementation of the software, this multiprocessor system can be used in a SPMD, a MPMD, or a hybrid configuration. A SPMD configuration can be considered if the Sparc is used by itself. Since the processors are connected via the VME bus, the interconnection network between processors can also be dynamically configured. Thus, this multiprocessor system allows a large degree of flexibility in designing and implementing an algorithm.

Two SPSD architectures for 3D backprojection are shown in Figure 5.4. For both architectures, each stage must be completed before proceeding to the next. At each stage, both architectures process a single view completely before proceeding to the next view. Thus only one input view needs to be retained in memory for processing. However, the two are different in their overall I/O and memory requirements. The architecture in Figure 5.4a requires only enough memory in each stage to hold one image volume if the missing projections in the second stage are output to disk. If the missing projections are retained in memory, the requirement increases significantly. For example, the missings projections for a typical 3D backprojection for the EXACT HR require about 15MB while an image volume of dimensions $47 \times 128 \times 128$ requires about 3MB. In contrast, the architecture in Figure 5.4b requires enough memory in the second stage to hold two image volumes, about 6MB. But there is no need to write the missing projections to disk since they are processed with the input view.



(a) SPSD architecture with single Image volume



(b) SPSD architecture with both Image volumes

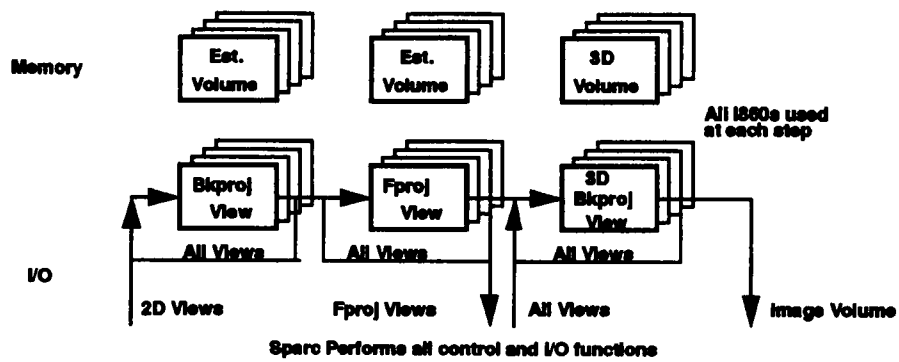
Figure 5.4. SPSD architectures for 3D backprojection.

From both of the above architectures, it is evident that first stage of the processing, the backprojection of the estimated image for forward-projection, **must** be completed before proceeding to the next stage. However, the processing within each stage is inherently parallel. Namely, the backprojection and forward-projection processes have a high degree of data parallelism since each view can be processed separately from the others. The only requirement is that the partial results from the backprojection process of each view be summed together at the end.

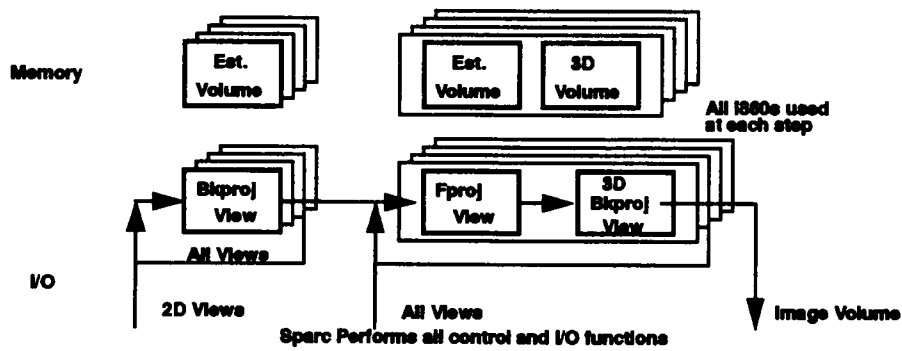
The second SPSP architecture indicates a possible opportunity for pipelining between the forward-projection and 3D backprojection step. However, load balancing may be difficult since the number of projections to be forward-projected varies based on axial angle. A more subtle opportunity for pipelining exists in inputting the views for processing. The views are typically contained in a file accessible by the Sparc, either a local or network disk, and must be read into memory accessible by the i860's. By using two buffers in each i860's memory, a set of new views can be loaded while another set are being processed.

Using the two SPSP architectures as references and the above discussion, various parallel/pipelined architectures for 3D backprojection on the above multiprocessor system may be described and analyzed. Figure 5.5 shows these architectures.

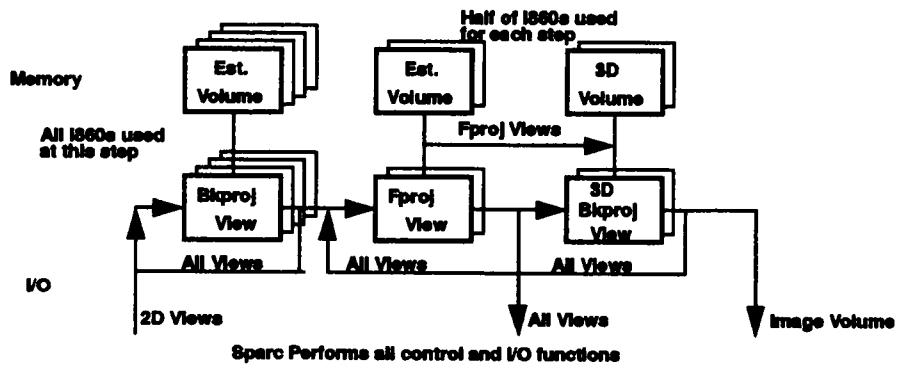
A SPMD architecture is shown in Figure 5.5a. This architecture, without the data pipelining, is the typical architecture used in most other parallel implementations of 3D backprojection [28]. A potential bottleneck for this architecture, particularly without data pipelining, is the I/O rate to/from disk. There are also two implementation variations related to how the image volume is divided among the i860's. One approach was mentioned above. Each i860 has a local copy of the full image volume and, when



(a) SPMD architecture with single Image volume



(b) SPMD architecture with both Image volumes



(c) SPMD/MPMD architecture with single Image volume

Figure 5.5. Parallel/pipelined architectures for 3D backprojection.

processing is complete, the volumes are added together. This approach encounters a problem when the image volume is larger than the memory available. Another approach divides the image volume across two or more i860's and, when processing is complete, the separate pieces are output to form the volume. Thus, the size of the image volume is limited by the total memory on all the i860's. However, the same set of views must be given to multiple i860's for each to correctly process the data. Thus, the communication overhead may increase, possibly reducing the efficiency.

Figure 5.5b shows another SPMD architecture which is very similar to the previous architecture. The major difference is that the forward-projection is done as a view is processed. Thus, the potential for having an I/O bottleneck is reduced. However, the amount of memory required is twice as large per i860 or the size of the image volume must be half as large. The two implementation variations presented above also apply, but the forward-projections must also be given to multiple i860's. Thus, a communications bottleneck may occur.

A hybrid SPMD/MPMD architecture is shown in Figure 5.5c. The first stage of this architecture is the same as the previous SPMD architectures. The second stage is similar to the second SPMD architecture except that the tasks of forward-projection and backprojection are given to different i860's. Thus the second stage is a MPMD architecture. The memory requirement on each i860 is reduced to that in the first SPMD implementation. However, the communication overhead increases and a load balancing problem is created.

Before comparing the above architectures, two other implementations of 3D back-projection on other multiprocessor systems should be presented. Both implementations use an algorithm other than the reprojection algorithm. However, both have 3D back-

projection as one of the stages in their algorithms. Thus, a common point of reference for comparison is possible.

The first was implemented by Chen et al. [11] on a 32-node hypercube, iPSC/2, and uses the Extended True Three-Dimensional Reconstruction (ETTR) algorithm [14]. This algorithm uses a special set of filters in an attempt to avoid doing a forward-projection step. The sequence of steps are a convolution followed by backprojection. The backprojection is performed using the 3D Incremental Algorithm [13]. This algorithm is a searching algorithm which reduces the operations involved in backprojection to only additions at each voxel in an attempt to eliminate floating point multiplications from the process.

Chen describes different parallelization and task partitioning approaches. The approach which was implemented is a MPMD architecture where the convolutions are performed by a set of nodes and the result is transmitted to another set of nodes which perform the backprojection. This produces a communications bottleneck and a load balancing problem. The load balancing problem is addressed somewhat by iterating the distribution of processing. The nodes are also limited in memory which results in a maximum image size which can be reconstructed using this MPMD architecture.

Table 5.1 has the implementation results achieved by Chen for an image size of $95 \times 95 \times 95$ which is approximately the same size image as for the EXACT HR above. Note that not all nodes were utilized. The "speedup" for this implementation was defined as the ratio of the sequential processing time to the parallel processing time with no statement as to how these were determined. The "efficiency" was defined as the speedup divided by the number of nodes used. Based on these definitions, the best

Table 5.1. Hypercube implementation results (Adapted from [11]).

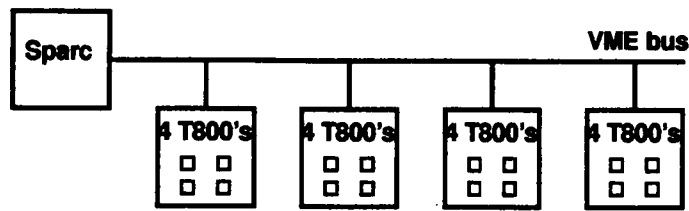
Image Size	# of Processors	Speedup	Efficiency
95x95x95	24	22.28	92.82
95x95x95	25	23.06	92.24
95x95x95	26	23.96	92.16

efficiency was 92.82%. It is evident from the conclusions by Chen that load balancing was the major difficulty and focus of their implementation.

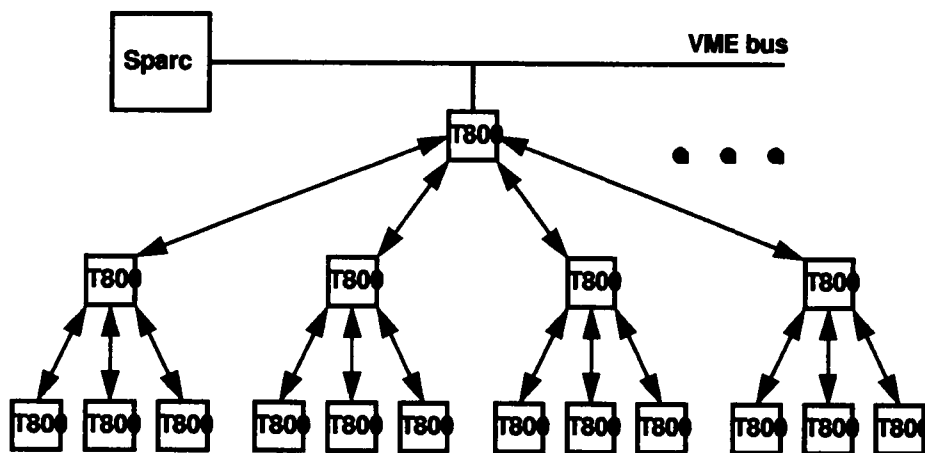
The other architecture which should be presented was implemented by Atkins et al. [2] on a network of T800 transputers with a Sparc as control unit. Two topologies were investigated and are shown in Figure 5.6. Similar to Chen, a convolution/backprojection algorithm is used to avoid the forward-projection. However, Atkins notes that they concentrated on speeding up the backprojection since 80% of their reconstruction time is spent doing backprojection.

Atkins notes that the topology shown in Figure 5.6a does not make use of any of the inter-transputer links available but treats each transputer as a separate processing node. This results a larger communication overhead which reduces the speedup and efficiency. Atkins uses definitions similar to Chen and reports a speedup of 13.3 for 16 transputers which is an efficiency of 83.125% using this topology and similar image sizes.

The topology shown in Figure 5.6b uses nodes consisting of one transputer with several transputers connected hierachically. This topology uses the communications ability of the transputers to improve performance. Atkins reports a speedup of 18 for



(a) Transputer topology with no inter-transputer links



(b) Hierarchical transputer topology with inter-transputer links

Figure 5.6. Transputer architectures for 3D backprojection (Adapted from [2]).

20 transputers which is an efficiency of 90.0% using this topology and similar image sizes as above. Atkins also estimates that 200 T800 transputers would be required to do the reconstruction in less than 10 minutes.

Similar to the coincidence detection and crystal identification, comparing all of the architectures presented above and determining which should be used for a given implementation can become a tremendous optimization problem. Hwang [34] describes the isoefficiency concept for scalability analysis by Kumar and Rao as one means of comparing algorithms and architectures. However in an effort to illustrate the similarities between parallel/pipelined ASIC designs and parallel/pipelined multiprocessor algorithms and systems, the optimization criteria for the 3D backprojection architectures are chosen to be area and performance.

In comparing the architectures, the assumption is made that the problem size is fixed and that the time is fixed. This correlates with the assumptions made for the parallel/pipelined ASIC designs. The "area" of this multiprocessor system can be related to one of two things: number of processors or memory per processor. The number of processors yields no useful information when comparing architectures with the same number of processors and fixed problem size. However, the memory required per processor by the architectures does yield useful information. The memory required per processor gives insight into the relative problem size which the architecture can handle for a fixed processor memory size.

As with the ASIC designs, the performance measure is much more nebulous. Using efficiency as the performance measure is not entirely valid since a single processor system has an efficiency of 1.0. However by basing the performance on the workload and overhead per time unit for a fixed time model, a representation of the architectures'

resource utilization can be found. To understand this, consider an ideal implementation of an algorithm on a multiprocessor system. The ideal efficiency would be 1.0 with n processors. For a single processor system to execute this algorithm in the *same* time, it would have to be n times faster than one of the n processors in the multiprocessor system. Thus the workload per time for a single processor system is n , where n is the number of processors in the multiprocessor system. On the other hand, the ideal multiprocessor system would have a workload per time of 1.0.

Similarly, the overhead is related to I/O and communications. With a fixed time model, i.e a pipelined model, the same data must be transferred to the processors but the *rate* will vary between architectures. Any architectures which require communications between processors would also exhibit a difference in communications rate for a fixed time model. By combining these three rates, workload rate, I/O rate, and communications rate, architecture differences relative to the multiprocessor system can be measured.

Table 5.2 lists the architectures and assigns each a letter. Table 5.3 contains the criteria values estimated for each architecture where n is the number of processors in the multiprocessor system. The limiting constraint for each architecture is also provided. The generation of the estimated image for forward projection is the same for all the architectures and does not need to be considered when producing the criteria values. The memory size is relative to that needed for a single processor. The performance rates are normalized to the non-zero minimums as in the ASIC designs. This table provides a basis for comparing the architectures when the trade-off between area and performance are the same, i.e. the designer feels that both are equally important. But a clearer

Table 5.2. 3D backprojection architectures.

Architecture	Identification
SPSD with single image volume	A
SPSD with both image volumes	B
SPMD with single local image volume	C
SPMD with single distributed image volume	D
SPMD with both image volumes local	E
SPMD with both image volumes distributed	F
SPMD/MPMD with local image volumes	G
SPMD/MPMD with distributed image volumes	H

Table 5.3. 3D backprojection architectural criteria values.

Architecture	Memory	Performance	Limiting Constraint
A	1	$n + 2$	Processor and memory speed
B	2	$n + 1$	Processor and memory speed
C	1	3.6	I/O speed
D	$1/n$	5.2	I/O and communication speed
E	2	2	Volume Size
F	$2/n$	4	Communication speed
G	1	4	Communication speed
H	$1/n$	5	Communication speed

Table 5.4. Criteria values for an 8 processor.

Architecture	Memory	Performance
A	1	10
B	2	9
C	1	3.6
D	0.125	8.8
E	2	2
F	0.25	4
G	1	4
H	0.125	5

understanding of the trade-offs between the architectures is possible with a plot where the trade-off between area and performance are not the same.

Table 5.4 shows the criteria values for the multiprocessor system described above for the EXACT HR with 8 i860s. Note that the criteria values are normalized to the minimum area and performance values, respectively. By normalizing the values to the minimums, the merit sums for each architecture are scaled relative to each other. Thus an architecture which requires twice the memory as the smallest architecture will have a weighted sum of 2.0 when memory is most critical.

Using Equations 3.1 and 3.2, a measure of an architecture's performance can be found at a particular trade-off point. By plotting the merit sums versus $\log(w_{a,i}/w_{p,i})$ for various trade-off points, a better comparison of the architectures is possible. Figure 5.7 is a plot of the merit sums for each architecture over a range of weight ratios

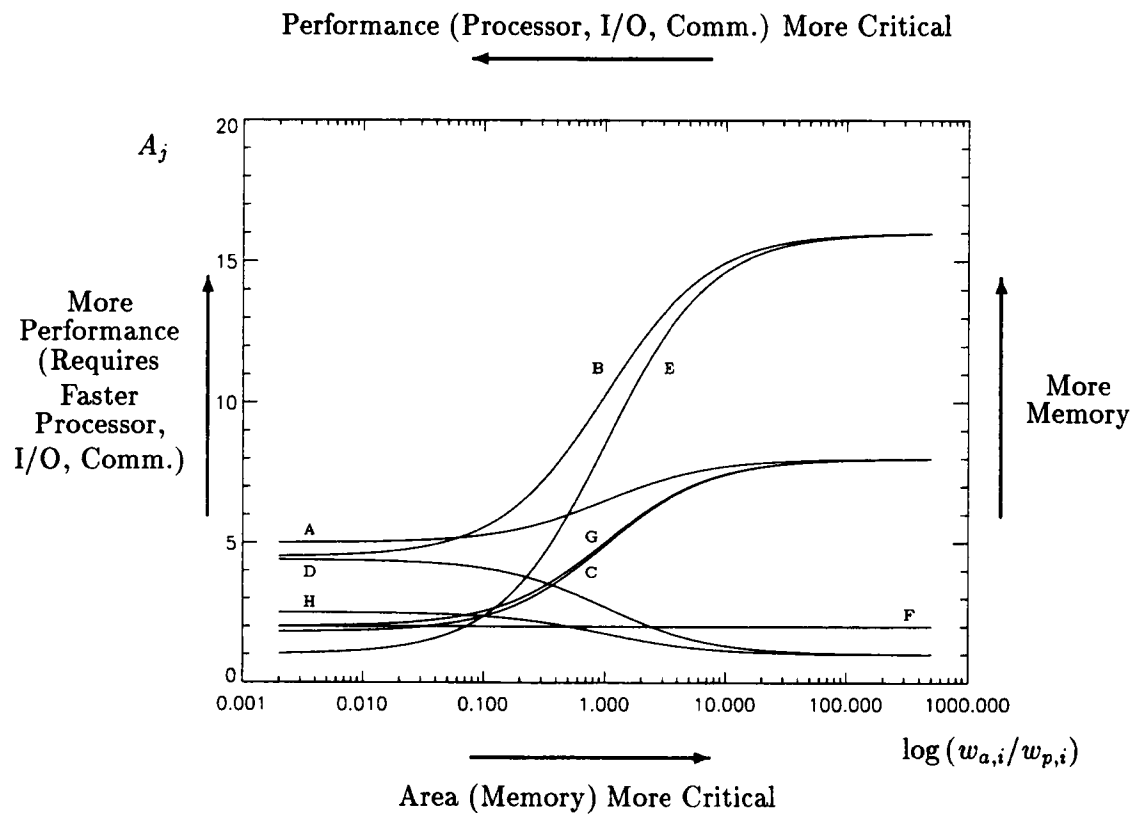


Figure 5.7. Plot of merit sums versus log of area/performance weight ratio.

from 0.001 to 1000.0. At a ratio of 1.0, the area and performance are deemed equally important. As one moves to the left to a ratio of 0.001, the performance is deemed more important than the area. As one moves to the right to a ratio of 1000.0, the area is deemed more important than the performance. Note, as one would expect, that the curves asymptotically approach the respective criteria values when that criteria dominates the decision.

From the plot, it is clear that architecture *E*, SPMD with both image volumes available in each processor, has the best performance and thus the best efficiency of the multiprocessor architectures. However, it requires processors with larger memories thus limiting the problem size. On the other hand, architectures *D* and *H* have the lowest memory requirements, but require higher performance rates. In particular, architecture *H* appears quite reasonable overall, but that ignores the fact that balancing the load between the processors could be difficult. Architecture *G* also suffers this handicap.

As with the ASIC designs, the architectures which have more parallel features perform better when performance is most important. Whereas, the architectures which have more serial features perform better when area (memory) is most important. However, the architectures which have flatter plots represent an inherently more equal trade-off between area and performance within the architecture. This is represented here by architecture *F* which has the flattest merit sums of any of the architectures.

5.3 Design and Implementation

The 3D backprojection software was designed and implemented in 1993-1994 for the multiprocessor system shown in Figure 5.3. The software consists of approximately

18,000 lines of C code of which 9,000 reside in each i860. The code for both the Sparc and the i860s was compiled using an optimizing compiler. The routines for backprojection and forward projection were hand-optimized for the i860 by the commercial vendor for CTI.

As with the ASIC designs, the above architectural analysis had not been fully developed at the time of this design. Therefore, a more heuristic approach was taken. Knowing that I/O and communications can account for a reduction in system efficiency, a parallel/pipelined architecture which had the least amount of communication between processors needed to be chosen. Also each i860 had 16MB of available memory which was enough to hold both image volumes for an EXACT HR ($47 \times 128 \times 128$ voxels per volume).

Thus, a SPMD architecture using both volumes in each i860 was chosen, essentially architecture *E* above. Other architectures were considered including architectures *C* and *G*. Architecture *C* was rejected because *E* performed better and the memory reduction was not needed at the typical volume size. Architecture *G* was rejected due to the load balancing requirements. Architecture *E* does not require load balancing since the projections can be evenly distributed among the processors which operate on them independently. Figure 5.8 shows a state diagram of the 3D backprojection software.

The state diagram contains several blocks which were not discussed in the above architectural analysis. These blocks are routines which correct the measured PET data for normalization and scatter correction. A complete description of the states and the state flow is presented in the CTI design document for the 3D backprojection software [55].

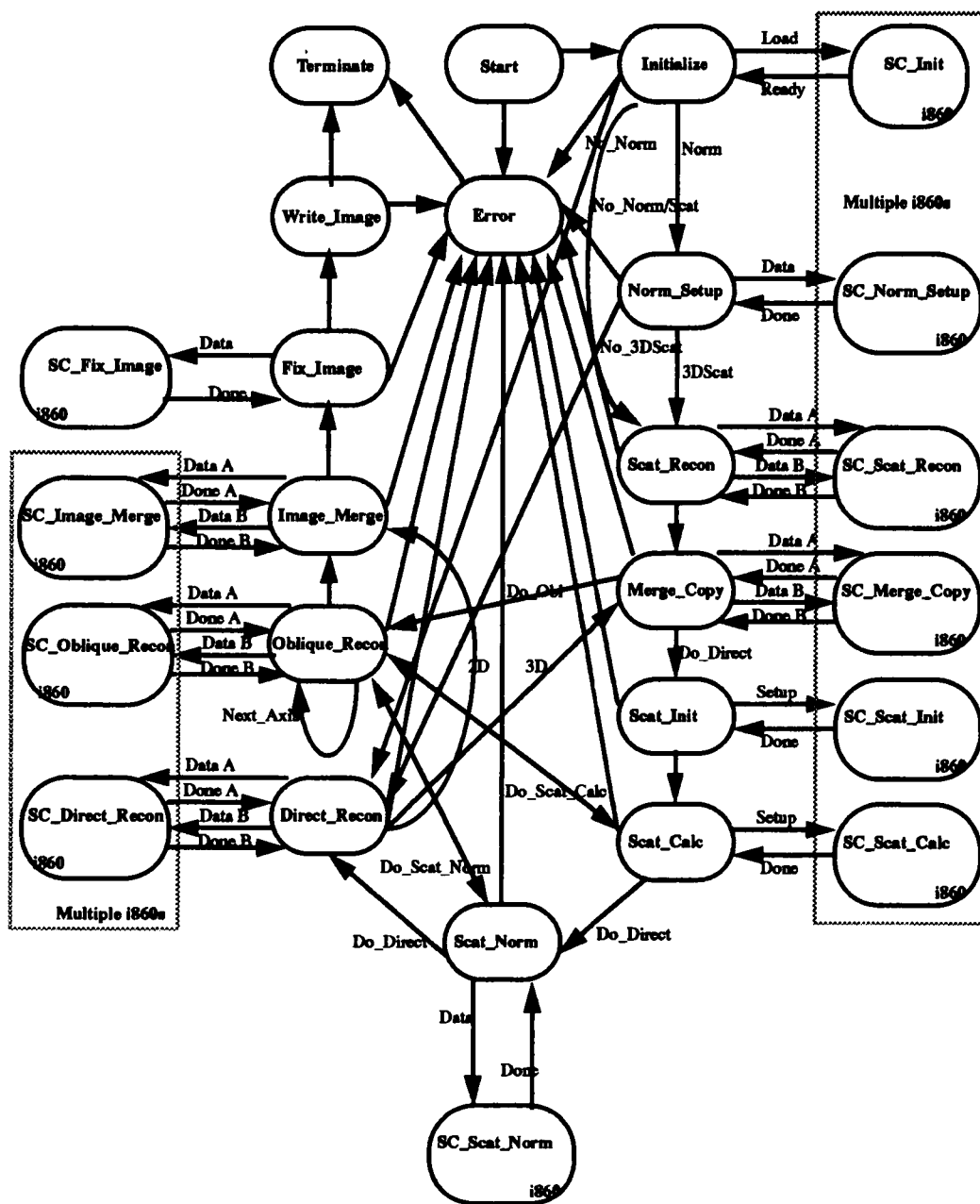


Figure 5.8. 3D backprojection state diagram.

Table 5.5. 3D backprojection execution times for 1 to 12 i860s.

# of Processors	Local Disk	Network Disk
	secs (mins)	secs (mins)
1	1907.6 (31.8)	1917.8 (32.0)
2	965.3 (16.1)	974.9 (16.2)
3	659.0 (11.0)	669.1 (11.2)
4	496.1 (8.3)	507.2 (8.5)
5	410.8 (6.8)	420.0 (7.0)
6	347.3 (5.8)	355.8 (5.9)
7	298.8 (5.0)	307.8 (5.1)
8	270.0 (4.5)	281.3 (4.7)
9	241.2 (4.0)	253.2 (4.2)
10	224.3 (3.7)	235.8 (3.9)
11	207.0 (3.4)	218.4 (3.6)
12	198.8 (3.3)	210.5 (3.5)

Table 5.5 lists the execution time for the 3D backprojection software using from one to twelve i860s. The times are averaged over five consecutive runs. The data set was from an EXACT HR reconstructed with no scatter correction into a $47 \times 128 \times 128$ volume. The two sets of times represent the input/output data sets being on a local disk versus a network disk. The major reason for the slowdown on the network disk is the writing of the image volume at the end of reconstruction. For a $47 \times 128 \times 128$ volume,

Table 5.6. 3D backprojection execution speedup and efficiency for 1 to 12 i860s.

# of Processors	Local Disk		Network Disk	
	Speedup	Efficiency (%)	Speedup	Efficiency (%)
1	1.00	100.0	1.00	100.0
2	1.98	98.8	1.97	98.4
3	2.89	96.5	2.87	95.5
4	3.85	96.1	3.78	94.5
5	4.64	92.9	4.57	91.3
6	5.49	91.5	5.39	89.8
7	6.38	91.2	6.23	89.0
8	7.07	88.3	6.82	85.2
9	7.91	87.9	7.57	84.2
10	8.50	85.0	8.13	81.3
11	9.22	83.8	8.78	79.8
12	9.59	79.9	9.11	75.9

a time of approximately four seconds is required to write the image volume to a local disk, whereas approximately 14 seconds are required to write to a network disk.

Table 5.6 contains the speedup and measured efficiency for the times shown in Table 5.5. Plots of the speedup and measured efficiency are shown in Figure 5.9 and Figure 5.10, respectively. Note that the curves are not linear, but follow Amdahl's Law which implies that the sequential portion of the program becomes dominant as more processors are added. The speedup can be related to the sequential portion of

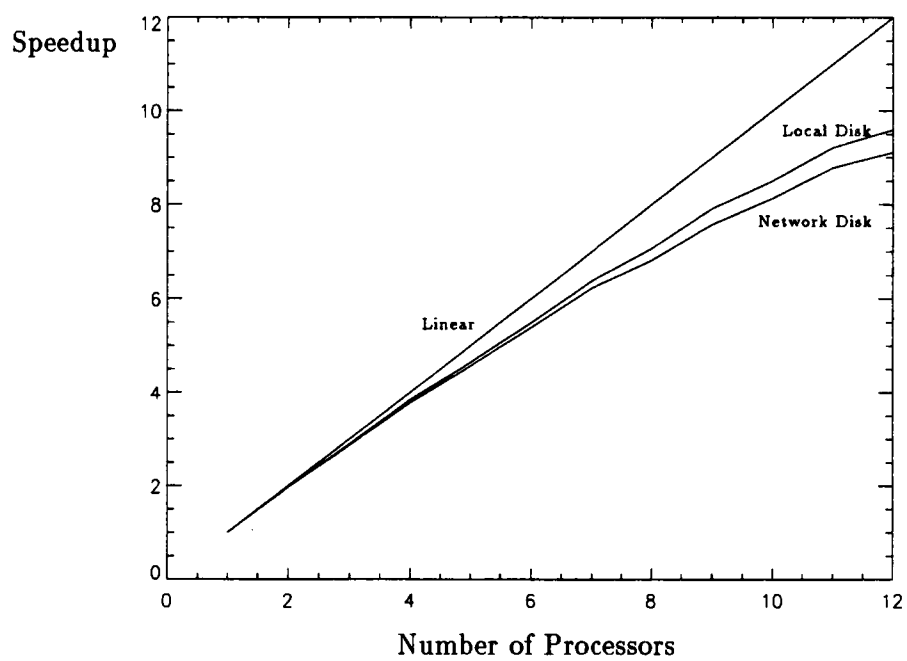


Figure 5.9. Speedup of 3D backprojection versus number of processors.

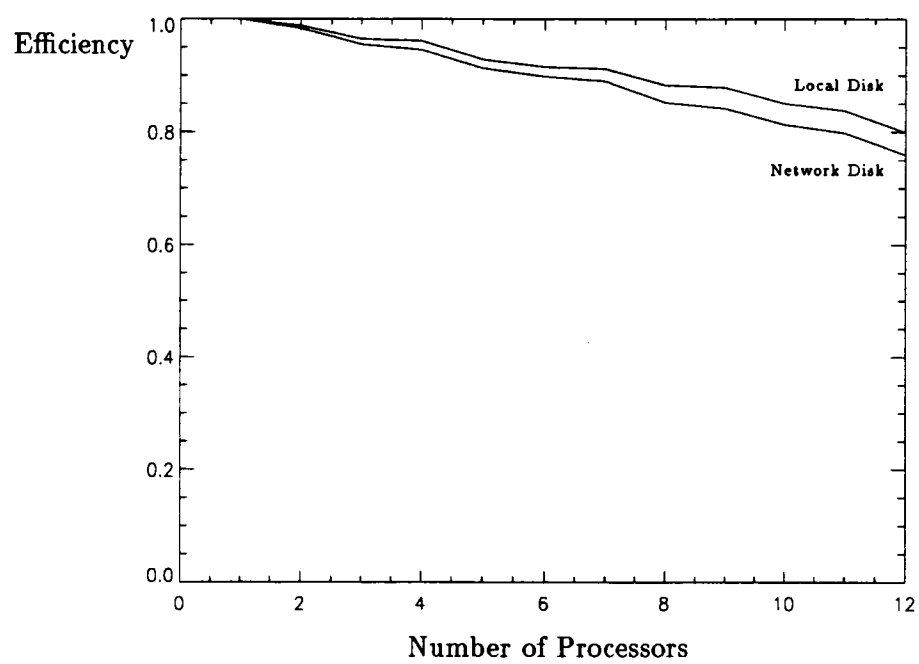


Figure 5.10. Efficiency of 3D backprojection versus number of processors.

the program and the number of processors by Equation 2.3 which is repeated here for convenience.

$$S(n) = \frac{n}{1 + (n - 1)\alpha}$$

where $S(n)$ is the speedup, n is the number of processors, and α is the sequential fraction of the program. Rearranging Equation 2.3

$$\alpha = \frac{\frac{n}{S(n)} - 1.0}{n - 1} \quad (5.1)$$

the sequential fraction of the program can be found from the speedup. Table 5.7 contains the sequential fractions found for Table 5.6. Note that the sequential fraction increases as the number of processors increases. The reason for this is that the Sparc has more processors to service in terms of I/O and communication. The Sparc's ability to service the processors becomes a limiting factor. However, the maximum sequential fraction is less 3% with 12 processors which is very good.

The standard configuration for an EXACT HR uses eight i860s producing an image volume in less than 5 minutes with no attenuation and scatter correction. Other implementations of the 3D backprojection have required significantly longer to run. For instance, implementations on Sparc workstations have taken on the order of 2 hours to execute the same size data set. As indicated by Atkins [2], upwards of 200 transputers would be required to have a reconstruction time of less than 10 minutes.

As an example of the need to reduce the reconstruction time, the author underwent a voluntary 3D dynamic brain scan while visiting the PET center at the Université Catholique de Louvain in Louvain-la-Neuve, Belgium. A dynamic brain scan is a set of scans performed sequentially in time from the start of an injection. The first 10 frames of the study are taken at one minute intervals and a remaining 5 frames are

Table 5.7. Sequential fraction for 2 to 12 i860s.

# of Processors	Sequential Fraction	
	Local Disk	Network Disk
2	1.20	1.67
3	1.82	2.34
4	1.34	1.93
5	1.92	2.38
6	1.85	2.26
7	1.61	2.06
8	1.89	2.48
9	1.72	2.35
10	1.96	2.55
11	1.94	2.52
12	2.28	2.88

taken at 10 minute intervals. The study allows the uptake of FDG into the brain to be monitored. Thus, a total of 15 frames of scan data have been taken. This means that a total of 15 reconstructions must be performed. If the reconstruction time was 1 hour per frame, a total of 15 hours would be required to reconstruct the full data set. With a reconstruction time of less than 10 minutes, the entire study can be reconstructed in 2.5 hours which allows a larger number of patients to be scanned. Figure 5.11 is a single slice of the author's study taken from the final 10 minute frame. Please note that there *is* activity present in his brain, contrary to rumors. Coregistration with an MRI scan would show the activity being concentrated on the surface and folds of the brain.

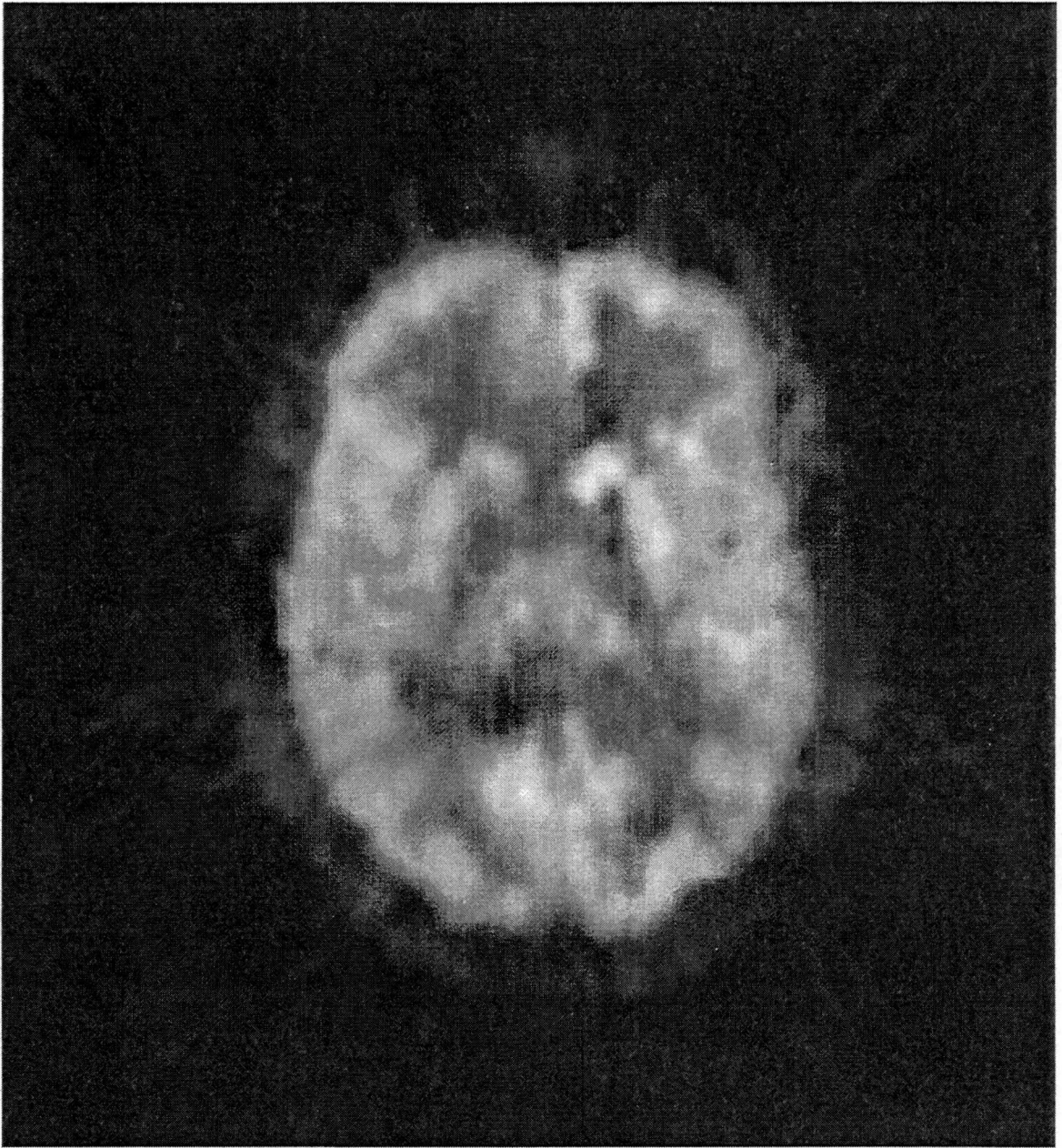


Figure 5.11. Single image slice from the author's 3D brain scan.

CHAPTER 6

Design Methodology for Parallel/Pipelined Systems

6.1 Description of the Problem

The preceeding chapters may appear to some to have no common basis. However, there is a thread which weaves through all of them and that is the design and implementation of parallel/pipelined systems. Parallel/pipelined systems, if properly implemented, can provide the most equitable trade-off between fully serial implementations (small area, more power) and fully parallel implementations (greater area, less power).

One of the greatest difficulties in using a parallel/pipelined design is choosing an architecture to achieve a “near optimal” trade-off between area and performance (power), if one exists. Analysis of potential architectures *before* a major design effort has been undertaken enables a designer to concentrate on the important aspects of the design at the appropriate places, i.e. a top-down design approach. This does not mean ignoring lower level design issues. On the contrary, the designer of parallel/pipelined systems should be even more of what Carver Mead describes as a “tall-thin” designer [48]. The designer must have an understanding of the effects a decision at the architectural level will have on lower levels of the design. Conversely, an understanding of the impact of lower level components on architectural concepts is required.

Designing parallel/pipelined systems requires not only breadth (tall) and some depth (thin, the “thicker” the better), but also an awareness of the spatial and temporal aspects of the architectures. A more parallel architecture tends to require more area (spatial) but less time (temporal). Whereas, a more serial architecture is the opposite, less area (spatial) but more time (temporal). Additional stages of pipelining requires an increased understanding of the temporal nature of the architectures.

All of the above may seem overwhelming to a designer challenged with a problem which could possibly benefit from a parallel/pipelined design. Thus, the purpose of this work. The designs presented in the preceeding chapters, in particular the ASIC designs, provide a point of study for understanding the concepts required for designing parallel/pipelined systems. An analysis of those designs should yield similarities and a basis for a design methodology.

6.2 Analysis of Designs of Parallel/Pipelined Systems

As each design was presented, some similarities with the other designs were pointed out. Some of those similarities may have been specific characteristics with limited application to other designs. The desire is to identify similarities in the designs which would influence the design methodology applied to problems beyond those presented here. Consequently, a step-by-step analysis of the design process considering all the designs simultaneously is appropriate.

Figure 6.1 shows the typical abstraction levels in system design. A variety of differing representations of these levels can be found in the literature [19, 63, 41]. The first step in a design is the problem description. The problem description for each design was

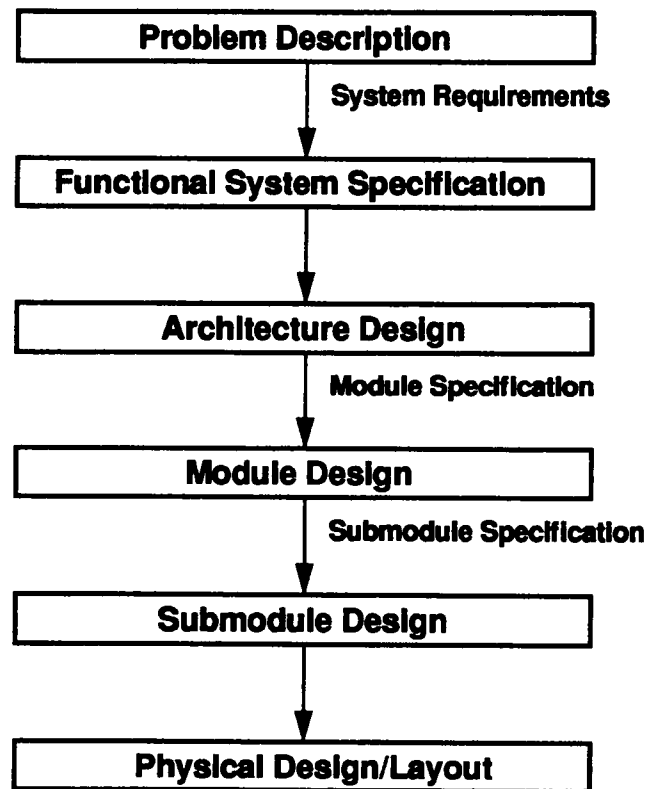


Figure 6.1. Levels of system design.

presented in the respective chapter. From the problem descriptions of the ASIC designs, the most notable aspect is that there is *no* timing requirement for when an output based on a specific set of inputs is to occur, i.e. the output at this clock cycle does not have to be the result of the current inputs. The only timing requirement is the cycle time of the inputs and outputs. This reveals an opportunity for pipelining. For 3D backprojection, noting that forward projection can occur independently of backprojection and that data can be input while other data is being processed offers a pipelining opportunity. Thus all three designs exhibit the potential for temporal parallelism.

From the problem description of the coincidence detection ASIC, there is no dependency *between* the time comparisons during a given cycle. The first generation ASIC required only the *notification* of a valid coincidence between the different comparisons. Likewise for the crystal identification ASIC, there is no dependency between crystal position comparisons. The problem description for 3D backprojection also leads to the conclusion that the computation of the partial sums as well as the forward projections can each be performed independently at different angles at the same time. Thus all three designs exhibit the potential for spatial parallelism.

From the problem description *and* an understanding of lower level implementation possibilities, all three of the designs presented exhibit the potential for spatial and temporal parallelism. In Chapter 2, the parallelism profile [34] of an algorithm was defined. The above analysis is a similar process which gives useful information when applied to the problem description.

The next step is the design of the functional system specification. In most cases, a “sequential” system specification can be defined and used to increase the level of understanding about the problem and possible implementations. For parallel software,

some [64] would contend that, in general, starting with a sequential algorithm and then developing a parallel version is not appropriate. However, using a sequential approach to gain a better understanding of the problem *is* appropriate and can lead to insights when developing the parallel form.

A case in point is the coincidence detection ASIC. During the design of the functional system specification for the first generation ASIC, the the question "Are the multiples *really* necessary?" was asked. The reason for the question was a consideration of a lower level issue, *communication between comparison elements* and an attempt to eliminate the necessity for having that communication during the design of the functional system specification. The answer at the time was "Yes." Two years later and a better understanding of the problem, the answer became "No, multiples are *not* necessary." Thus, the second generation coincidence detection ASIC had a different functional system specification. This specification did not require communication between comparison elements which increased the number of possible architectures that could be implemented.

A similar situation occurred during the same stage of the design for the crystal identification ASIC. Initially, the understanding of the problem description lead to a requirement that the "shape" of the crystal boundaries could be anything. The existing RAM-based solution in essence permitted this. The question "Is any shape *really* necessary? Will rectangles work?" was asked. The answer was that rectangles were acceptable and that the boundaries being loaded into the existing system were already rectangles. Again the reason for the question was a realization that the requirement to allow boundaries of any shape would produce complications at a *lower* level in the design.

The questions for the two designs occurred while developing a sequential functional system specification for the designs in an effort to better understand the problems. However, the questions might not have occurred if the impact to lower levels of the designs had not been considered.

After establishing a functional system specification, the next step is to design the implementation architecture. This is the step which was analyzed in the preceeding chapters on the designs. It is the step at which the designer can influence the resulting implementation the greatest. Using the information derived during the problem description stage concerning spatial and temporal parallelism, multiple implementation architectures can be proposed. These architectures should be based on pipelined stages and multiprocessor-like architectures, i.e. SPSD, SPMD, MPSD, MPMD. Hybrid architectures should also be considered since many problems are not necessarily fully serial or fully parallel [9, 64, 43, 60]. Also, hybrid architectures are often the ones which offer a "better" area/performance trade-off, but are often the most difficult for designers to quantify in terms of area/performance.

Choosing the implementation architecture to pursue is the step which requires the most understanding of the relationship between an architecture and implementation details at lower levels. This is clearly evident in the designs presented here. Being able to abstract those implementation details and make use of them in determining a "near optimal" architecture is a key element of designing parallel/pipelined systems.

As initially presented in the analysis of the coincidence detection architectures, Chapter 3, the area and performance of the different architectures can be used as a means to choose the appropriate architecture. The typical approach is to plot the area versus the performance for the various architectures for a given problem[19] with the

area and performance considered to be equally important to the design. The resulting set of points yields information *only* at that particular trade-off point. If the designer feels that area is more critical than performance, the result has to be inferred from the plot. The approach used in the analysis of the designs presented here is to give “weights” to the values for area and performance based on their importance to the designer. Thus, a measure of an architecture’s performance, which will be called the *merit sum*, at a particular trade-off point is given by Equations 3.1 and 3.2 which are repeated here for convenience

$$w_{a,i} + w_{p,i} = 1.0$$

$$A_{j,i} = w_{a,i}C_{a,j} + w_{p,i}C_{p,j}$$

where $w_{a,i}$ and $w_{p,i}$ are area and performance weights, respectively, for the trade-off point i , $A_{j,i}$ is the merit sum of architecture j at the trade-off point i , and $C_{a,j}$ and $C_{p,j}$ are the area and performance criteria values, respectively, for architecture j . The lower the merit sum, the “better” the architecture is at that trade-off point. Note that by normalizing the values to the minimum area and performance values, the merit sums for each architecture are scaled relative to each other. Thus an architecture which requires twice the area as the smallest architecture will have a merit sum of 2.0 when area is most critical.

By plotting the merit sums versus $\log(w_{a,i}/w_{p,i})$ for various trade-off points, a better comparison of the architectures is possible. Figure 6.2 is a plot of the merit sums for each coincidence detection architecture over a range of weight ratios from 0.001 to 1000.0. Note that for the ASIC designs the performance criteria chosen is power. At a ratio of 1.0, the area and power (performance) are deemed equally important. As one moves

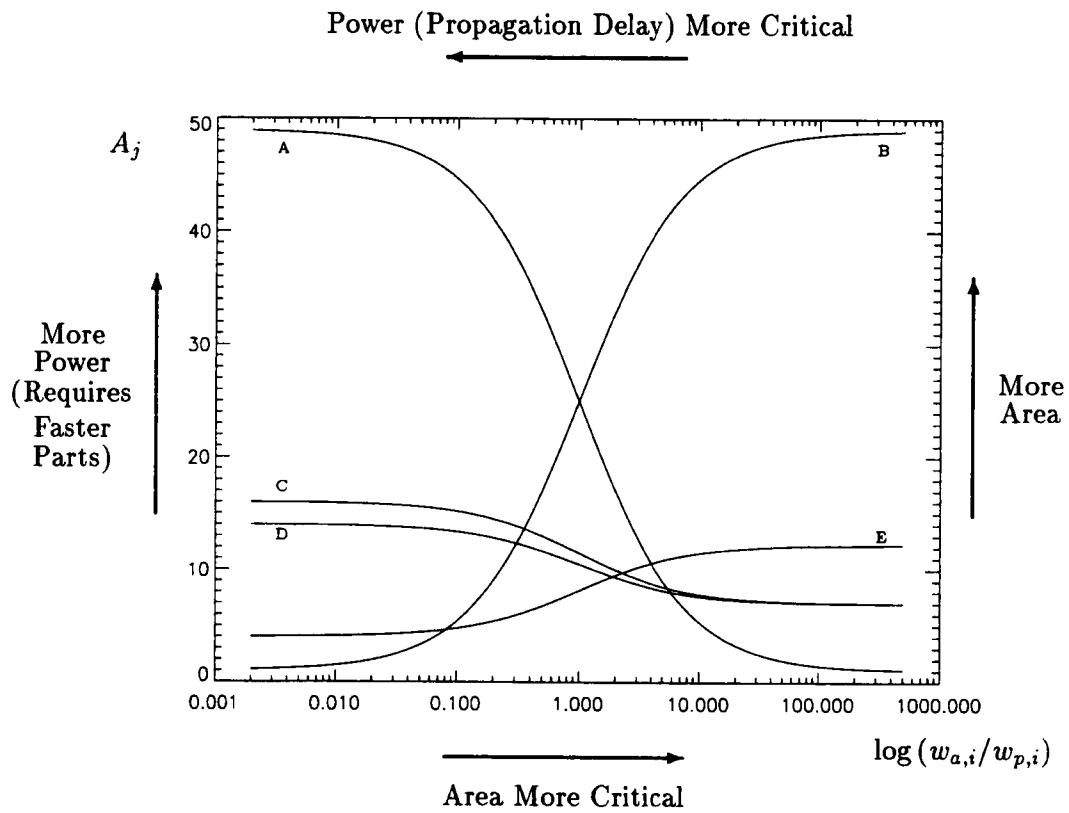


Figure 6.2. Coincidence detection merit sums versus log of area/power weight ratio.

to the left to a ratio of 0.001, the power is deemed more important than the area. As one moves to the right to a ratio of 1000.0, the area is deemed more important than the power. Note, as one would expect, that the curves asymptotically approach the respective criteria values when that criteria dominates the decision.

The use of this type of plot during the analysis of the designs presented here demonstrates its usefulness. The plot allows a designer to visualize the area/performance trade-offs between architectures which range from fully serial to fully parallel. In essence, the architectures which have more parallel features perform better when performance is most important. Whereas, the architectures which have more serial features perform better when area is most important. However, the architectures which have flatter plots represent an inherently more equal trade-off between area and power within the architecture. In many cases, the trade-off between area and power is not as large as plotted (1000 to 1) but tends to be small (1 to <10 or <10 to 1) and not well defined. For these, the architectures with flatter plots and lower means are the better overall performers.

Several points can be extracted from the analysis of the designs concerning abstracting lower level implementation details and using them to choose an architecture. The *relative* area and performance between key components is more important than the actual area and performance of specific components. Note the comparison blocks in the coincidence detection ASIC, the comparison blocks and RAM in the crystal identification ASIC, and memory, I/O, and communication in 3D backprojection. *Identifying* those key components is critical. Different architectures will require different optimizations of these key components at lower levels. Having an understanding of the *limits* to these optimizations for the implementation technology of choice is necessary. For example, the SPSP architecture for the coincidence detection ASIC required a comparison

block which could perform a comparison in approximately 5 nanoseconds. This is not feasible with any amount of optimization in 1.2 micron CMOS. However, in another technology, such as GaAs, or in CMOS technology in a few years, it might be feasible.

After an implementation architecture has been selected, the remaining steps in the design process tend to follow the typical system design philosophy. Optimizations at each implementation level are driven by the requirements set by the architecture chosen *and* the implementation technology. For example, the architecture for the first generation coincidence detection ASIC was chosen based on performance, but at a penalty in area. Thus the comparison blocks were optimized for area since there was “performance to burn” but area was a premium. Similarly, the second generation ASIC architecture was chosen as a more equitable trade-off between area and performance. However, the implementation technology, 1.2 micron CMOS gate array, required the comparison blocks to be optimized for performance (speed).

Note that throughout the design and implementation process, any aspect of the designs which appeared to introduce a bottleneck to the designs were treated as “opportunities” for improvement. The retargeting of an ASIC design to different vendors described in Chapter 4 is an improvement to the design approach driven by a potential vendor “bottleneck.” The modeling of the ASIC functionality in a high level language presented in the same chapter is an improvement to the design approach to aid in overcoming the test vector bottleneck. It also aided in understanding and verifying the design.

6.3 Description of Design Methodology

Figure 6.3 diagrams a methodology for designing parallel/pipelined systems. Note that the methodology is an expansion of the system design process in Figure 6.1. Certain aspects of the methodology pertain to all steps and need to be stated. Understanding the problem cannot be overstated. Each step appears as a separate functional block, however the designer should be aware of and be considering how each step affects the others (tall-thin designer). Bottlenecks, wherever they occur, either in the design approach or at an implementation stage, should be viewed as "opportunities" for improvement in the design not as problems to be avoided.

Given a problem description, the designer needs to determine if there is potential for implementing the system as a parallel/pipelined system. This is accomplished by doing a parallelism profile of the problem description. The designer should look for potential areas for temporal and spatial parallelism. Potential areas for temporal parallelism are those for which no "timing" relationship between input and output data exists, the potential for an item to be processed while the result from the previous time is being processed, or input/output of data can occur while other data is processed. Potential areas for spatial parallelism are processes, identical or different, which can be performed independently of one another on the same or different data.

If there is potential for designing a parallel/pipelined system, the designer produces a functional system specification. While doing this design, the designer should not be afraid to ask questions concerning the requirements of the system. A simplification of the system architecture could result from an alternative problem description. Questions of the type presented in the above analysis of the designs are examples. The identification

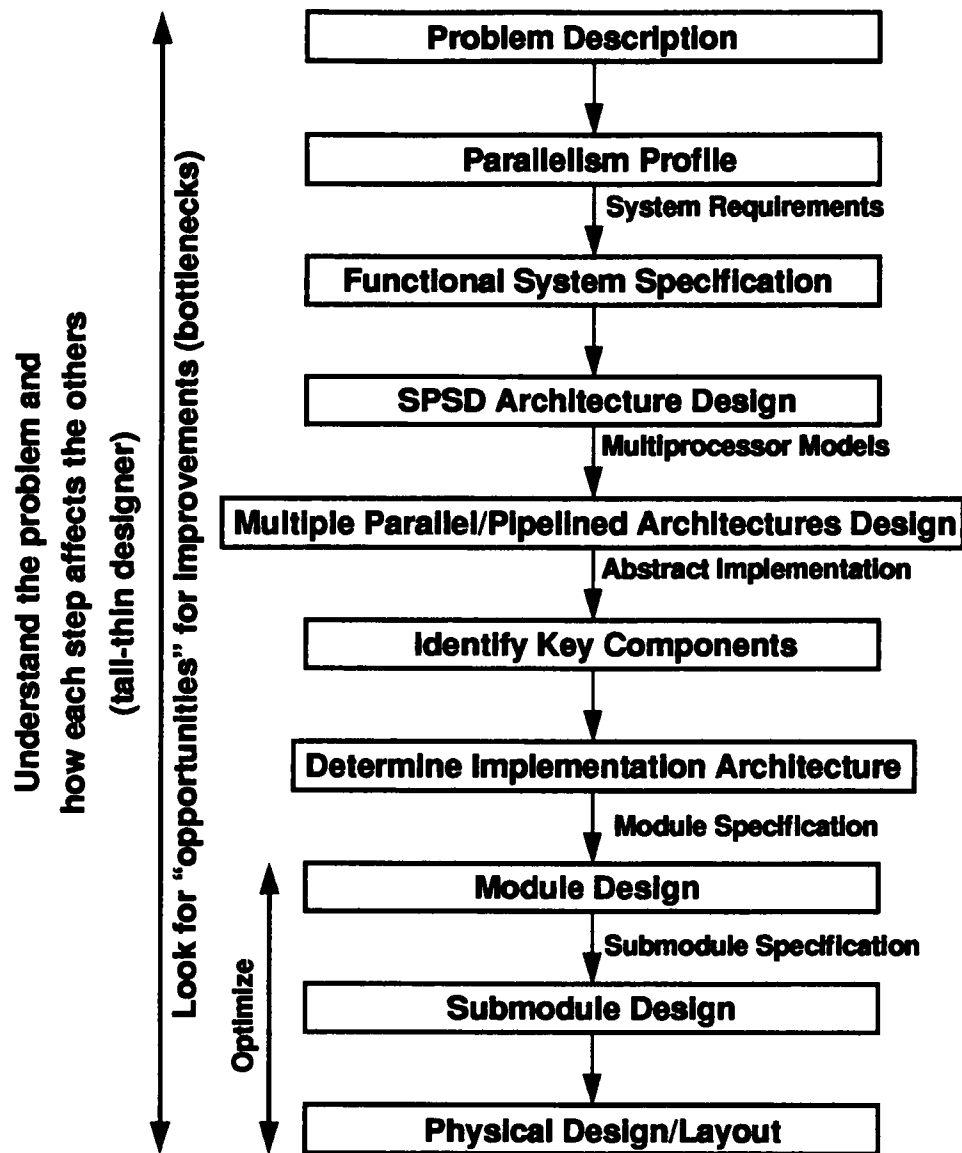


Figure 6.3. Parallel/pipelined design methodology.

of input/output data pipelines should also occur at this stage. The designer should be aware that the design of inputs/outputs are driven by the packaging and communications technology chosen at the implementation level.

Based on the functional system specification, a SPSD architecture can be designed. The goal of this stage is to develop a better understanding of the system requirements that are driven by the specification. The identification of potential temporal and spatial parallelism should also occur at this stage.

After analyzing the SPSD architecture, the designer should use multiprocessor models (SPSD, SPMD, MPSD, MPMD) and/or hybrid models to design parallel/pipelined architectures to meet the functional system specification. Identification of key components which drive the area and performance of the architectures should occur during the design of the architectures. The relative area and performance required by the key components for each architecture should also be determined.

The relative area and performance characteristics are used as criteria values for the architectures. These criteria values are used in Equation 3.2 to generate the merit sums for each architecture. These merit sums are then plotted as discussed above. From the plot, the designer can identify the architecture which best suits the overall requirements of the system. Considerations should be given at this point to limitations (speed, area, power, etc.) in implementing a given architecture. The "more optimum" architectures in terms of a trade-off between area and performance are those with flatter and lower curves.

With an architecture chosen, module and submodule design is performed followed by physical design and layout. Optimizations required by the given architecture and implementation technology are performed during these stages. Techniques and computer-

aided design tools to optimize various aspects of the design at these stages are prevalent throughout the literature [19, 58, 3, 40].

The physical design and layout stage is determined by the implementation technology. If the technology is a CMOS gate array, the physical design and layout is typically performed by a commercial vendor with *placement and critical path* information provided by the designer. The designer derives this information from the lowest level of the design while considering the architectural form of the design. In many cases, the architecture itself forms a placement map of the design. For example, Figure 6.4 shows the placement of the modules for the crystal identification ASIC in a CMOS gate array. Note the similarity to the architecture shown in Figure 4.4b.

If possible, the designer should design the system as vendor-independent as possible. Designing the system to be technology independent is possible, but one of the design criteria is that the implementation technology influences design decisions throughout the *whole* design process. Thus, a technology-independent design would not be as well suited for some technologies as others. Also for ASIC designs in particular, modeling the problem and the chosen architecture in a high level language can increase the understanding of the problem and aid in testing. If possible, the designer should consider this in addition to the design methodology.

One final word on architectures and designing is appropriate for the designer considering a parallel/pipelined design for the first time: “System architecture and design, like art, can only be learned by doing”, Carver Mead and Lynn Conway [49].

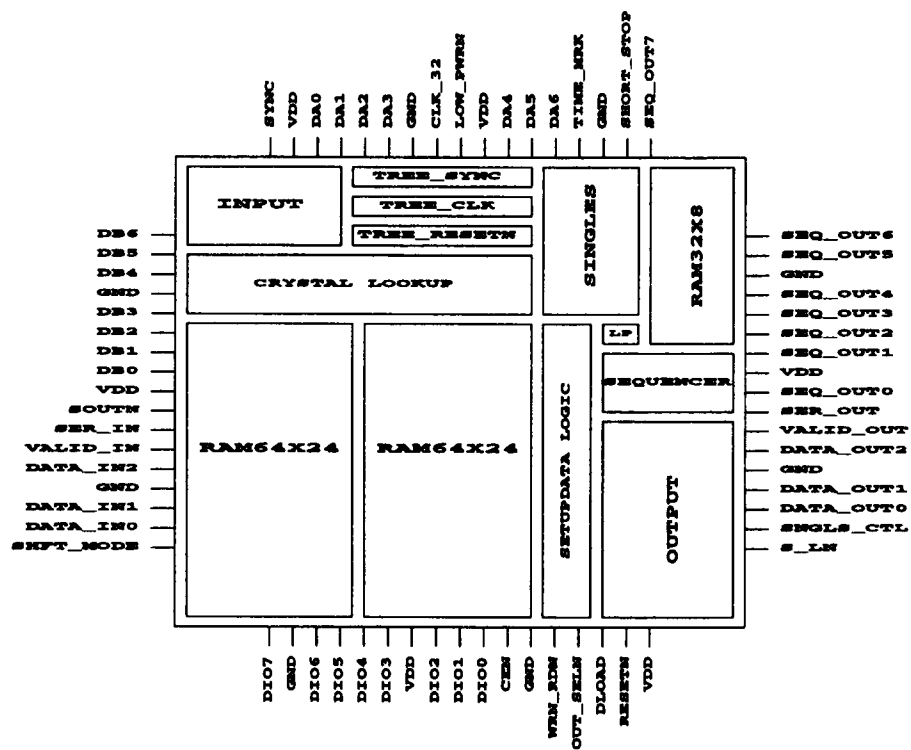


Figure 6.4. Floorplan for crystal identification ASIC.

CHAPTER 7

Concluding Remarks

7.1 Summary

The design and analysis of three parallel/pipelined systems were presented. From the analysis of the designs, an understanding of the design process flow was established. With this understanding, a methodology for designing parallel/pipelined systems, particularly ASIC systems, was presented. The methodology stresses the use of a “tall-thin” design approach where the designer has an understanding of the effects of each design stage on the other stages.

An experienced designer, using the methodology, can determine early in the design process whether a problem can be solved using a parallel/pipelined system. Multiprocessor models were used to describe and design possible architectures. Analysis of the architectures based on the implementation technology yield relative area/performance information. The methodology incorporates area/performance curves to compare the architectures.

Parallel/pipelined systems require the designer to spend more design time and effort at the early stages of the design than a serial design. However using the methodology, a designer will be able to understand what is needed for the design effort and what architectures to pursue.

The methodology was emphasized for ASIC systems. However, the third design was a multiprocessor software design. The analysis of this design indicated similarities between it and the two ASIC designs. This lends support to the possible use of this methodology for multiprocessor software design. Support for the similar design strategies in software and VLSI can be found in Smith [63].

Design of reconfigurable systems for specific problems could possibly use this methodology. An empirical methodology for reconfigurable architectures has been presented by Ligon[46]. The main use of reconfigurable systems is for parallel/pipelined systems in an effort to improve performance. A methodology, such as that presented here, could aid in overcoming the design hurdles for reconfigurable systems [5].

Similarly, *IEEE Design & Test of Computers* devoted almost an entire issue to hardware-software codesign, particularly methodologies [35]. Since there was a mixture of hardware and software designs which lead to this methodology, there is the potential for this methodology to be useful in hardware-software codesign.

7.2 Future Work

The design methodology presented in this work is an attempt to describe a process by which a parallel/pipelined system can be designed. The methodology for designing such systems will continue to evolve as more designers pursue such systems and the design process is better understood. Formalization and incorporation of the methodology into a design tool could potentially increase a designer's productivity.

Research in reconfigurable architectures for parallel/pipelined systems is in its infancy. Application of this methodology, or a variation, to reconfigurable architectures

could aid the designer in designing architectures which yield improvements in performance while achieving a reduction in area.

In hardware-software codesigns, the hardware and software tend to operate in a parallel/pipelined fashion. Thus, a variation of this methodology could aid a designer in partitioning a design into the appropriate hardware and software components for a more efficient design.

LIST OF REFERENCES

LIST OF REFERENCES

- [1] A. Antola, A. Avai, and L. Breveglieri, "Modular design methodologies for image processing architectures", *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 1, no. 4, pp. 408-414, December 1993.
- [2] M. S. Atkins, D. Murray, and R. Harrop, "Use of transputers in a 3-d positron emission tomography", *IEEE Transactions on Medical Imaging*, vol. 10, no. 3, pp. 276-283, September 1991.
- [3] R. A. Bergamaschi, R. Camposano, and M. Payer, "Data-path synthesis using path analysis, *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pp. 591-596, 1991.
- [4] R. Bianchini and J. Shen, "Interprocessor traffic scheduling algorithm for multiple - processor networks", *IEEE Transactions on Computers*, vol. C-36, 4, pp. 396-409, 1987.
- [5] C. Brown, "Reconfigurable computing hits snags", *Electronic Engineering Times*, issue 871, pp. 91-94, October 1995.
- [6] M. Broy, "Towards a design methodology for distributed systems, M. Broy, editor, *Constructive Methods in Computing Science*, Springer-Verlag, Berlin, DE, 1989, NATO ASI Series Vol. F55.
- [7] N. Carriero and D. Gelernter, "How to write parallel programs: A guide to the perplexed", *ACM Computing Surveys*, vol. 21, no. 3, pp. 323-357, September 1989.
- [8] M. E. Casey, *An Analysis of Counting Losses in Positron Emission Tomography*, Ph.D. Dissertation, University of Tennessee, Knoxville, Electrical Engineering Department, December 1992.
- [9] F. Catthoor, "Microcoded processor architectures and synthesis methodologies for real-time signal processing, E. F. Deprettere and A.-J. van der Veen, editors, *Algorithms and Parallel VLSI Architectures Volume A: Tutorials*, pp. 403-429, Elsevier Science Publishers, 1991.
- [10] S. Ceri and S. Navathe, "A methodology for the distribution design of databases, *Proceedings COMPCON Spring '83*, San Francisco, February 1983.
- [11] C. M. Chen, C. M. Chen, and Z. H. Cho, "A parallel implementation of 3-d ct image reconstruction on hypercube multiprocessor", *IEEE Transactions on Nuclear Science*, vol. 37, no. 3, pp. 1333-1346, June 1990.
- [12] M. C. Chen, "A design methodology for synthesizing parallel algorithms and architectures", *Journal of Parallel and Distributed Computing*, vol. 3, pp. 461-491, 1986.

- [13] Z. H. Cho, C. M. Chen, and S.-Y. Lee, "Incremental algorithm - a new fast back-projection scheme for parallel beam geometries", *IEEE Transactions on Medical Imaging*, vol. 9, no. 2, pp. 207-217, June 1990.
- [14] Z. H. Cho, J. B. Ra, and S. K. Hilal, "True three-dimensional reconstruction (ttr) - application of algorithm toward full utilization of oblique rays", *IEEE Transactions on Medical Imaging*, vol. MI-2, no. 1, pp. 6-18, March 1983.
- [15] A. N. Choudhary and J. H. Patel, *Parallel Architectures and Parallel Algorithms for Integrated Vision Systems*, Kluwer Academic Publishers, Boston, 1990.
- [16] J. G. Colsher, "Fully three-dimensional positron emission tomography", *Physics in Medicine and Biology*, vol. 25, no. 1, pp. 103-115, 1980.
- [17] M. E. Conway, "A multiprocessor system design, *Proceedings of the AFIPS Fall Joint Computer Conference*, pp. 139-146, 1963.
- [18] J. Darlington, M. Reeve, and S. Wright, "Programming parallel computer systems using functional languages and program transformation, *Parallel Processing '89*, Leiden, The Netherlands, 1989.
- [19] G. De Micheli, *Synthesis and Optimization of Digital Circuits*, McGraw-Hill, Inc., New York, 1994.
- [20] J.-M. Delosme and I. Ipsen, "Efficient systolic arrays for the solution of Toeplitz systems: An illustration of a methodology for the construction of systolic architectures in VLSI, *Systolic Arrays*, pp. 37-46, Adam Hilger, Ltd., Bristol, 1987.
- [21] J. B. Dennis and E. C. Van Horn, "Programming semantics for multi-programmed computations", *Comm ACM*, vol. 9, pp. 143-155, 1966.
- [22] H. M. Dent, W. F. Jones, and M. E. Casey, "A real-time digital coincidence processor for positron emission tomography", *IEEE Transactions on Nuclear Science*, vol. 33, no. 1, pp. 556-559, February 1986.
- [23] E. W. Dijkstra, "Co-operating sequential processes, F. Genyus, editor, *Programming Languages*, pp. 43-112, Academic Press, 1967.
- [24] M. J. Flynn, "Some computer organizations and their effectiveness", *IEEE Transactions on Computers*, vol. 21, no. 9, pp. 948-960, 1972.
- [25] R. L. Geiger, P. E. Allen, and N. R. Strader, *VLSI Design Techniques for Analog and Digital Circuits*, McGraw-Hill Publishing Company, New York, 1990.
- [26] A. D. George and L. W. Hawkes, *MICROPROCESSOR-BASED PARALLEL ARCHITECTURE for Reliable Digital Signal Processing Systems*, CRC Press, Boca Raton, 1992.

- [27] I. Gorton, J. Gray, and I. Jelly, "PARSE: A methodology for parallel program design, J. Hulskamp and D. Jones, editors, *Transputers and Parallel Applications*, pp. 101-106, Amsterdam, 1993, IOS Press.
- [28] T. M. Guerrero, S. R. Cherry, M. Dahlbom, A. R. Ricci, and E. J. Hoffman, "Fast implementations of 3d pet reconstruction using vector and parallel programming techniques, *Proceedings of the 1992 IEEE Nuclear Science Symposium and Medical Imaging Conference*, pp. 969-971, 1992.
- [29] E. L. Hall, *Computer Image Processing and Recognition*, Academic Press, New York, 1979.
- [30] J. Hennessy and D. Patterson, *Computer Architecture: A Quantitative Approach*, Morgan Kaufman, San Mateo, CA, 1990.
- [31] M. P. Herlihy, "A methodology for constructing highly concurrent data structures, *ACM Symposium on Principles and Practice of Parallel Programming*, March 1990.
- [32] G. T. Herman, *Image Reconstruction from Projections - The Fundamentals of Computerized Tomography*, Academic Press, New York, 1980.
- [33] P. Hudak and L. Smith, "Para-functional programming: A paradigm for programming multiprocessor systems, *Thirteenth Annual ACM Symposium on Principles of Programming Languages*, pp. 243-254, St. Petersburg Beach, Florida, January 15-16, 1986, ACM Press, New York.
- [34] K. Hwang, *Advanced Computer Architecture: Parallelism, Scalability, and Programmability*, McGraw-Hill, Inc., New York, 1993.
- [35] *IEEE Design & Test of Computers*, vol. 10, no. 3, September 1993.
- [36] K. Irani and N. Khabbaz, "A methodology for the design of communication networks and the distribution of data in distributed supercomputer systems", *IEEE Transactions on Computers*, vol. C-31, 5, pp. 419-434, 1982.
- [37] A. Kalavade and E. A. Lee, "A hardware/software codesign methodology for dsp application", *IEEE Design & Test of Computers*, vol. 10, no. 3, pp. 15-28, September 1993.
- [38] A. Kapelnikov, R. R. Muntz, and M. D. Ercegovic, "A methodology for performance analysis of parallel computations with looping constructs", *Journal of Parallel and Distributed Computing*, vol. 14, no. 2, pp. 105-120, February 1992.
- [39] P. E. Kinahan and J. G. Rogers, "Analytic three-dimensional image reconstruction using all detected events", *IEEE Transactions on Nuclear Science*, vol. 36, no. 1, pp. 964-968, February 1989.

- [40] D. W. Knapp, "An interactive tool for register-level structure optimization, *Proceedings of the 26th ACM/IEEE Design Automation Conference*, pp. 598-601, 1989.
- [41] R. Kober and W. Wenderoth, "Problems and practical experience in high-level design, *Proceedings of the 1983 International Conference on Computer Design: VLSI in Computers & Processors*, pp. 427-430, 1983.
- [42] S. Kong, D. Wood, G. Gibson, R. Katy, and D. Patterson, "Design methodology for a vlsi multiprocessor workstation", *VLSI Systems Design*, vol. x, pp. 44-54, February 1987.
- [43] E. V. Krishnamurthy, *PARALLEL PROCESSING: Principles and Practice*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1989.
- [44] H. T. Kung, "Let's design algorithms for VLSI systems, *Proceedings of the Caltech Conference on VLSI*, pp. 65-90, 1979.
- [45] H. T. Kung and C. E. Leiserson, "Systolic arrays (for VLSI), *Sparse Matrix Proceedings*, 1978.
- [46] W. B. Ligon III and U. Ramachandran, "An empirical methodology for exploring reconfigurable architectures", *Journal of Parallel and Distributed Computing*, vol. 19, no. 4, pp. 323-337, December 1993.
- [47] D. J. Lilja, editor, *Architectural Alternatives for Exploiting Parallelism*, IEEE Computer Society Press, Los Alamitos, California, 1991.
- [48] C. Mead, *Lectures on VLSI Design*, 1985.
- [49] C. Mead and L. Conway, *Introduction to VLSI Systems*, Addison-Wesley Publishing Company, Reading, Massachusetts, 1980.
- [50] D. A. Menasce and L. A. Barosso, "A methodology for performance evaluation of parallel applications on multiprocessors", *Journal of Parallel and Distributed Computing*, vol. 14, no. 1, pp. 1-14, January 1992.
- [51] P. M. Merlin, "A methodology for the design and implementation of communication protocols", *IEEE Trans. Comm.*, vol. COM-24, pp. 614-621, June 1976.
- [52] J. D. Mertens and W. L. Bhend, "Digital coincidence detection: A scanning VLSI implementation, *Proceedings of the 1992 IEEE Nuclear Science Symposium and Medical Imaging Conference*, pp. 879-881, 1992.
- [53] J. C. Moyers, *A High Performance Detector Electronics System for Positron Emission Tomography*, Master of Science Thesis, University of Tennessee, Knoxville, Electrical Engineering Department, May 1990.

- [54] W. G. Nation, A. A. Maciejewski, and H. J. Siegel, "A methodology for exploiting concurrency among independent tasks in partitionable parallel processing systems", *Journal of Parallel and Distributed Computing*, vol. 19, no. 3, pp. 271-278, November 1993.
- [55] D. F. Newport, bkproj_3D Design Document, CTI, Inc., Knoxville, TN, 1995.
- [56] D. F. Newport, H. M. Dent, M. E. Casey, and D. W. Bouldin, "Coincidence detection and selection in positron emission tomography using VLSI", *IEEE Transactions on Nuclear Science*, vol. 36, no. 1, pp. 1052-1055, February 1989.
- [57] D. F. Newport and J. W. Young, "An ASIC implementation of digital front-end electronics for a high resolution PET scanner", *IEEE Transactions on Nuclear Science*, vol. 40, no. 4, pp. 1017-1019, August 1993.
- [58] A. C. Parker, P. Gupta, and A. Hussain, "The effects of physical design characteristics on the area-performance tradeoff curve, *Proceedings of the 28th ACM/IEEE Design Automation Conference*, pp. 530-534, 1991.
- [59] M. E. Phelps, J. C. Mazziotta, and H. R. Schelbert, editors, *Positron Emission Tomography and Autoradiography - Principles and Applications for the Brain and Heart*, Raven Press, New York, 1986.
- [60] J. H. Reif, editor, *Synthesis of Parallel Algorithms*, Morgan Kaufmann Publishers, San Mateo, California, 1993.
- [61] J. G. Rogers, R. Harrop, and P. E. Kinahan, "The theory of three-dimensional image reconstruction for pet", *IEEE Transactions on Medical Imaging*, vol. 6, no. 3, pp. 239-243, September 1987.
- [62] S. R. Sarukkai, D. Kimelman, and L. Rudolph, "A methodology for visualizing performance of loosely synchronous programs", *Journal of Parallel and Distributed Computing*, vol. 18, no. 2, pp. 242-251, June 1993.
- [63] C. U. Smith and J. A. Dallen, "A comparison of design strategies for software and for vlsi, *Proceedings COMPCON Spring '83*, pp. 263-268, 1983.
- [64] J. R. Smith, editor, *The Design and Analysis of Parallel Algorithms*, Oxford University Press, New York, 1993.
- [65] C. R. Snow, *Concurrent Programming*, Cambridge University Press, Cambridge, 1992.
- [66] E. E. Swartzlander Jr., editor, *Systolic Signal Processing Systems*, Marcel Dekker, Inc., New York, 1987.
- [67] D. E. Thomas, J. K. Adams, and H. Schmit, "A model and methodology for hardware/software codesign", *IEEE Design & Test of Computers*, vol. 10, no. 3, pp. 6-15, September 1993.

- [68] A. Trew and G. Wilson, editors, *PAST, PRESENT, PARALLEL: A Survey of Available Parallel Computer Systems*, Springer-Verlag, London, 1991.
- [69] P. Tseng, *A Systolic Array Parallelizing Compiler*, Kluwer Academic Publishers, Boston, 1990.
- [70] J. W. Young, M. E. Casey, and D. F. Newport, "Optimum bandwidth usage in digital coincidence detection for PET, *Proceedings of the 1993 IEEE Nuclear Science Symposium and Medical Imaging Conference*, pp. 1205–1208, 1993.

VITA

Danny Fieldon Newport was born in LaFollette, Tennessee on August 14, 1957. After graduating in June 1975 as valedictorian from Jacksboro High School in Jacksboro, Tennessee, and a brief appearance at the United States Military Academy at West Point that summer, he enrolled in the College of Engineering at the Knoxville campus of the University of Tennessee. He participated in the Cooperative Engineering program and graduated with a Bachelor of Science Degree in Electrical Engineering with high honors in August 1981.

After receiving his B.S., he worked for Union Carbide as a Development Associate and began pursuit of his Master's Degree. He left Union Carbide in March 1984 and began work as the systems manager in the College of Engineering Computer Laboratory at the University of Tennessee. While employed there, he completed his Master of Science Degree in Electrical Engineering in December 1986.

After completing his M.S., he began working with the Oak Ridge National Laboratory (ORNL) in Oak Ridge, Tennessee through the University of Tennessee and consulting with CTI, Knoxville, TN in ASIC design. He also began working toward a PhD at the University of Tennessee. In January 1993, he began working full-time at CTI. He continues to enjoy a relationship with ORNL and the University.

He has been married to Rosemary Warren Newport since 1985. They have two children, Sarah Elizabeth and Daniel McFee (Mac), whom they love and cherish. They all share a home with two cats, Warlock and Kimberley, and a boxer, Lil Walt.