1

CANADIAN THESES ON MICROFICHE

I.S.B.N.

THESES CANADIENNES SUR MICROFICHE

National Library of Canada
Collections Development Branch

Canadian Theses on
Microfiche Service

Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Direction du développement des collections

Service des thèses canadiennes
sur microfiche

NOTICE

The quality of this microfiche is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us a poor photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this film is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30. Please read the authorization forms which accompany this thesis.

**THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED**

AVIS

La qualité de cette microfiche dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de mauvaise qualité.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, examens publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de ce microfilm est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30. Veuillez prendre connaissance des formules d'autorisation qui accompagnent cette thèse.

**LA THÈSE A ÉTÉ
MICROFILMÉE TELLE QUE
NOUS L'AVONS REÇUE**

National Library
of CanadaBibliothèque nationale
du Canada

Canadian Theses Division

Division des thèses canadiennes

Ottawa, Canada
K1A 0N4

59932

PERMISSION TO MICROFILM — AUTORISATION DE MICROFILMER

• Please print or type — Écrire en lettres moulées ou dactylographier

Full Name of Author — Nom complet de l'auteur

PETER ALEXANDER MANASHE

Date of Birth — Date de naissance

Country of Birth — Lieu de naissance

Permanent Address — Résidence fixe

P.O. BOX 2702, STN. "D", OTTAWA, ONT., CANADA.

Title of Thesis — Titre de la thèse

A NOVEL HIGH PERFORMANCE
DATA FLOW COMPUTER ARCHITECTURE.

University — Université

CARLETON UNIVERSITY, OTTAWA, ONT.

Degree for which thesis was presented — Grade pour lequel cette thèse fut présentée

MASTER OF ENGINEERING

Year this degree conferred — Année d'obtention de ce grade

1982

Name of Supervisor — Nom du directeur de thèse

B.A. BOWEN

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

Date

15. Sept. 82

Signature

Peter Manashe

**A NOVEL HIGH PERFORMANCE
DATA FLOW COMPUTER ARCHITECTURE**

by

© Peter A. Manashe

A thesis submitted to the
Faculty of Graduate Studies and Research
in partial fulfilment of the requirements for
the degree of
Master of Engineering

Carleton University
Ottawa, Ontario

August 1982

The undersigned recommend to the
Faculty of Graduate Studies and Research
acceptance of the thesis

**A NOVEL HIGH PERFORMANCE
DATA FLOW COMPUTER ARCHITECTURE**

submitted by
Peter A. Manashe
in partial fulfilment of the requirements for
the degree of
Master of Engineering

Thesis Supervisor

Chairman
Department of Systems
and Computer Engineering

Carleton University
Ottawa, Ontario

August 1982

ABSTRACT

This thesis developed an efficiency model for computer architectures. The efficiency model is applied to data flow computers in order to identify critical bottlenecks and suggest mechanisms to eliminate them. Digital signal processing is used as the algorithmic benchmark, because of its inherent requirement for high performance. A novel data flow computer is suggested that has significantly higher performance than contemporary architectures.

ACKNOWLEDGEMENTS

I would like to thank my supervisor Dr. Archie Bowen for his many valuable suggestions and guidance throughout this thesis, my wife Nicole for her infinite patience and Richmond Communications for expedient word processing of this thesis.

TABLE OF CONTENTS

PAGE

Abstract	i
Acknowledgements	ii
Table of Contents	iii
List of Figures	iv
List of Abbreviations	v

1. INTRODUCTION

1.0 The problem of efficient architectures	1.4
1.1 Data flow computers: classical architecture (adv/dis)	1.18
1.2 Limitations of DFC	1.25
1.3 The Thesis proposal and summary of results	1.30

2. ISSUES in the DESIGN of DATA FLOW COMPUTERS

2.0 Introduction	2.2
2.1 Architectural efficiency measures	2.2
2.2 System issues	2.7
2.3 Implementation issues	2.17
2.4 The myth about unlimited expandability of DFC's	2.19
2.5 Summary of design issues	2.21

3. A NOVEL HIGH PERFORMANCE DATA FLOW ARCHITECTURE

3.0 Introduction	3.2
3.1 Operational Requirements for DSP	3.2
3.1.1 Computational requirements	3.2
3.1.2 Throughput requirements	3.6
3.1.3 System control and synchronization	3.9
3.1.4 Architectural flexibility	3.10
3.2 A novel high performance data flow architecture	3.12
3.3 Summary	3.23

4. CONCLUSIONS

4.0 Introduction	4.2
4.1 Summary and conclusions	4.2
4.2 Topics for further research	4.3

5. APPENDIX

5.1 Application areas for the novel DFC architecture	5.2
5.2 Glossary of terms	5.4
5.3 References	5.5

LIST OF FIGURES

PAGE

Figure 1.01	Multiprocessor architectures	1.8
Figure 1.02	Centralized control	1.13
Figure 1.03	Pipelined control	1.15
Figure 1.04	The classical data flow architecture	1.19
Figure 1.05	An activity template	1.21
Figure 1.06	Interconnection of operators	1.22
Figure 1.07	Comparison of data flow architectures	1.28
Figure 2.1.1	Model for the data flow computer architecture	2.4
Figure 2.2.1	Centralized control	2.10
Figure 2.2.2	Fully distributed control	2.12
Figure 2.2.3	Data packets for example 2.3	2.16
Figure 3.2.1	Segmented memory architecture	3.13
Figure 3.2.2	Multiport memory architecture	3.15
Figure 3.2.3	The multiport memory	3.16

LIST OF ABBREVIATIONS

VLSI	...	Very Large Scale Integration
FFT	...	Fast Fourier Transform
MIPS	...	Millions of Instructions Per Second
A.E.	...	Architectural Efficiency
ALU	...	Arithmetic Logic Unit
DFC	...	Data Flow Computer
I/O	...	Input/Output
MIT	...	Massachusetts Institute of Technology
C.T.	...	Control Throughput
DSP	...	Digital Signal Processing
VTR	...	Video Tape Recorder
PE	...	Processing Element
C _N	...	Control Unit N
M _N	...	Memory Unit N
ART	...	Address Recognition Time
MAT	...	Memory Access Time
R/D	...	Research/Development
CAD	...	Computer Aided Design
CAM	...	Computer Aided Manufacturing

1.1

1. INTRODUCTION

1. INTRODUCTION

Many applications require the maximum computational performance that today's technology can offer. Among them are digital signal processing, weather forecasting, air traffic control and others.

The availability of low cost processors in the form of VLSI chips has sparked much interest in highly parallel computational engines. [31, 33, 35, 36, 37, 38]

Among these parallel architectures currently under investigation are vector computers [24, 34], systolic computers [30] and data flow computers [26, 32]. This thesis will be concerned with data flow architectures for highly parallel machines, in particular their performance, efficiency and ways of improving the architecture.

In a data flow machine (described more fully in section 1.1), the control section sends instruction and data packets to a multiplicity of processing units and in turn receives a packet of results from these processing units.

To achieve a maximum performance in a computer, the following three ingredients are needed:

1. Multiplicity of processing elements
2. High throughput of the control section

3. Decrease in the control overhead.

A multiplicity of low cost processing elements, together with their proper synchronization, is the key to the high performance of data flow computers.

High throughput of the control section is achieved by distribution of the control section among many processing cells as proposed by Dennis [26].

Reduction of control overheads is an important consideration in this thesis. This will be accomplished by the use of multiport memories, data communication strategy by address and extensive use of hardware macros. Hardware macros are VLSI chips that perform a high level algorithm without a need for detailed external control. Examples of such chips are array multipliers and chips for FFT's.

1.0 THE PROBLEM OF EFFICIENT ARCHITECTURES

Computational engines are normally optimized for speed. In many cases, little attention is paid to the fact that large amounts of computational resources have to be sacrificed for a small increase in the actual computational output.

Close examination of this problem reveals that the disproportionately small increases in computational output can be linked to what we will call the efficiency of the computer architecture.

Many varied figures of merit can be employed for comparison of different architectures. Among them are: computational throughput in millions of instructions per second (MIPS), throughput per unit cost (MIPS/\$), utilization in percent of maximum output, and others. All these figures of merit do not say anything about the internal efficiency of the computational engine on which this result was achieved.

The efficiency of a computer architecture can be expressed as the ratio of the computational effort used for production of useful output to the total computational effort spent. The total computational effort can be split into control overhead, used for internal housekeeping of the computational engine and the useful computational output. A new concept, the architectural efficiency (A.E.) of a computer is defined in Equation 1.2.

COMPUTATIONAL EFFORT E

$$E = \frac{\text{OUT} \times \text{UTIL}}{100} \quad (1.1)$$

OUT ... output in MIPS at 100% utilization

UTIL ... utilization in %

ARCHITECTURAL EFFICIENCY A.E.

$$\text{A.E.} = \frac{E_u}{E_u + E_c} \quad (1.2)$$

E_u ... net useful computation effort (output)

E_c ... control effort.

Example 1.1

A sequential machine with 100% utilization spends $E_u = 10$ MIPS on useful computational output and $E_c = 10$ MIPS on control. The architectural efficiency of the above machine is:

$$A.E. = \frac{E_u}{E_u + E_c} = \frac{10}{10 + 10} = 0.5 = 50\%$$

DISCUSSION

The architectural efficiency measure has the advantage of identifying bottlenecks and inefficiencies in the hardware architecture of a given computational engine. Therefore, by using the architectural efficiency measure, areas for improvement can be clearly identified.

The preceding definition of the architectural efficiency does not consider overlapping of execution and control times because the computational effort spent on control and execution is the same for overlapped and non-overlapped systems. Therefore, overlapping of control and execution times does not change the architectural efficiency of a given computational engine.

In order to maximize the efficiency of a computational engine, the control overhead should be brought as close to zero as possible.

To maximize the useful output of a computational engine at the lowest cost, both throughput and efficiency have to be optimized

at the hardware and software level. Note that efficiency must not be considered in isolation. A computer with a semiconductor control unit and a relay ALU would have a high efficiency but not a high throughput.

Example 1.2

A sequential machine with 100% utilization spends $E_u = 10$ MIPS on useful computational output and $E_c = 10$ MIPS on control. The execution of overhead and useful output are overlapped by 50%.

The architectural efficiency of the above machine is:

$$A.E. = \frac{E_u}{E_u + E_c} = \frac{10}{10 + 10} = 0.5 = 50\%$$

The following examples show the effect of architectural efficiency on computer architectures.

Example 1.3 Multiprocessor with individual controllers

Let a computational engine have the configuration shown in Fig. 1.01(a). Each of the four processing elements PE_1 through PE_4 is capable of producing $OUT_1 = OUT_2 = OUT_3 = OUT_4 = 10$ MIPS at a 100% utilization factor. Each control element $CONTR.1$ through $CONTR.4$ has a throughput of $COUT_1 = COUT_2 = COUT_3 = COUT_4 = 10$ MIPS at 100% utilization factor. Let the utilization be 50% for

FIGURE 1.01: MULTIPROCESSOR ARCHITECTURES

both the processing and control elements. Calculate the average throughput in MIPS and the architectural efficiency of this computational engine.

Solution

Throughput AV:

$$AV = UTIL (OUT_1 + OUT_2 + OUT_3 + OUT_4)$$

$$AV = 0.5 (10 + 10 + 10 + 10) = 20 \text{ MIPS}$$

architectural efficiency:

$$A.E. = \frac{\sum_{k=1}^4 UTIL_k \times OUT_k}{\sum_{k=1}^4 UTIL_k \times OUT_k + \sum_{k=1}^4 UTIL_k \times COUT_k}$$

$$A.E. = \frac{20}{20 + 20} = 0.5 = 50\%$$

EXAMPLE 1.4 The Vector Processor.

Let a computational engine have a configuration as in FIG. 1.01(b). Each of the four processing elements PE_1 through PE_4 is capable of a throughput of $OUT_1 = OUT_2 = OUT_3 = OUT_4 = 10 \text{ MIPS}$ when utilized 100%. The single control element CONTR. is capable of a throughput of $COUT = 10 \text{ MIPS}$ when utilized 100%. All the processing elements use the same instruction. Let the utilization factor be 50% for each of the processing and control elements.

1.10

Calculate the average throughput in MIPS and the architectural efficiency in percent.

Solution:

Throughput:

$$AV \doteq UTIL (OUT_1 = OUT_2 = OUT_3 = OUT_4)$$

$$AV = 0.5 (10 + 10 + 10 + 10) = 20 \text{ MIPS}$$

Architectural efficiency:

$$A.E. = \frac{\sum_{k=1}^4 UTIL_k \times OUT_k}{UTIL \times COUT + \sum_{k=1}^4 UTIL_k \times OUT_k}$$

$$A.E. = \frac{20}{5 + 20} = 0.8 = 80\%$$

EXAMPLE 1.5 The optimized vector processor.

Let a computational engine have the configuration shown in FIG 1.01(c). Each of the four processing elements PE_1 through PE_4 is capable of a throughput of $OUT_1 = OUT_2 = OUT_3 = OUT_4 = 10 \text{ MIPS}$ when utilized 100%. The single control element CONTR. has a throughput of $COUT = 1 \text{ MIPS}$ when utilized 100%. Let the utilization be 50% for both the processing and control elements. All processing elements use the same instruction. Calculate the average throughput in MIPS and the architectural efficiency of this computational engine.

Solution:

Throughput:

$$AV = UTIL (OUT_1 + OUT_2 + OUT_3 + OUT_4)$$

$$AV = 0.5 (10 + 10 + 10 + 10) = 20 \text{ MIPS}$$

Architectural efficiency:

$$A.E. = \frac{\sum_{k=1}^4 UTIL_k \times OUT_k}{UTIL \times COUT + \sum_{k=1}^4 UTIL_k \times OUT_k}$$

$$A.E. = \frac{20}{0.5 + 20} = 0.98 = 98\%$$

DISCUSSION

Examples 1.3, 1.4 and 1.5 show the value of architectural efficiency as a comparison criteria. Comparison of all three architectures on the basis of throughput or utilization will show them to be exactly equivalent. The comparison by architectural efficiency reveals that the architecture of FIG. 1.01(c) has the highest architectural efficiency. Higher architectural efficiency means lower control overhead per unit throughput in MIPS. This translates into a smaller and cheaper control unit and therefore a more cost effective architecture. If the PE's were multipliers, then one could say that example 1.4 used single word

multipliers while example 1.5 used array multipliers and therefore required less control for the same output.

THE CENTRALIZED CONTROL PROBLEM: Architectures with centralized control (Fig. 1.02) have one controller which sequences the operations $F_1 \dots F_N$. A good example of such an architecture is the widely used von Neuman machine [5, 7, 9, 18, 24].

Let a processor consist of:

$F [f_1 f_2 \dots f_N]$ operations, with

$E [e_1 e_2 \dots e_N]$ execution utilization and

$C [c_1 c_2 \dots c_N]$ control utilization.

Let each element of F be linearly mapped on E and C .

An algorithm consisting of a sequence of instructions from F , the elements of which are executed k_i times will have a total useful computational effort of:

$$E_{op} = \sum_{i=1}^N k_i e_i \quad (1.3)$$

Equation 1.3 is another way of stating equation 1.1. For each f_i , a control overhead is required. The total computational effort required for

FIGURE 1.02: CENTRALIZED CONTROL

control overhead for the algorithm is equal to:

$$C = \sum_{i=1}^N k_i c_i \quad (1.4)$$

Total computational effort for the algorithm is equal to the sum of useful computational effort and overhead:

$$RT = E_{op} + C = \sum_{i=1}^N k_i e_i + \sum_{i=1}^N k_i c_i \quad (1.5)$$

The efficiency can be expressed as:

$$A.E. = \frac{T_{op}}{C + E_{op}} = \frac{\sum_{i=1}^N k_i e_i}{\sum_{i=1}^N k_i c_i + \sum_{i=1}^N k_i e_i} \quad (1.6)$$

N ... number of operations

One way of achieving control multiplicity is the data flow computer [1, 12, 15, 26] which will be discussed next.

THE DATA FLOW COMPUTER PROBLEM: In a data flow computer, the centralized control of FIG. 1.02 is replaced by a pipelined controller (FIG. 1.03). Due to this pipeline style design of the data flow computer control section, the average control throughput is increased by a factor N,

FIGURE 1.03 : PIPELINED CONTROL

(the number of pipeline stages) under steady state operating conditions.

A typical data flow architecture has four pipeline stages: activity store, fetch, instruction queue and update which are more fully defined in section 1.1. For this type of architecture, the average control throughput would equal four times the throughput of the weakest link in the pipeline. The architectural efficiency of a data flow computer will equal:

$$\text{A.E.} = \frac{\sum_{i=1}^N k_i e_i}{\sum_{i=1}^N k_i c_i + \sum_{i=1}^N k_i e_i} \quad (1.7)$$

N ... number of operations.

By using a pipelined design for the control section, data flow computers speed up the processing of the controller. This leads to a higher processing capability of data flow computers, as compared to von Neuman architecture, and therefore higher throughput. This increase in processing speed can only be fully utilized in algorithms having inherent parallelism of processing.

Two fundamental problems limit [14] the processing capabilities of data flow computers:

- a) Finite number of stages in the control pipeline
- b) Congestion on buses due to improper data communication strategy.

DISTRIBUTED CONTROL: Distributed control implies equivalent multiplicity of control. As the control is distributed over M control units, the equivalent average throughput (AV) of the control section is significantly increased compared to the throughput of a single control unit (COUT). The throughput of a distributed control unit of multiplicity M equals:

$$AV = \sum_{k=1}^M COUT_k \times UTIL_k \quad (1.8)$$

$$A.E. = \frac{\sum_{i=1}^N k_i e_i}{\sum_{i=1}^N k_i c_i + \sum_{i=1}^N k_i e_i} \quad (1.9)$$

N ... number of operations

For a distributed control unit of multiplicity M , in which one control unit is assigned to each processing element, the limitations inherent in a data flow computer with a centralized control unit no longer apply.

The control throughput of the machine is significantly increased through the use of a large number of inexpensive control units.

The bus congestion problem is significantly reduced because most of the bus traffic is conducted over many local buses instead of a single main bus. The main bus is only used for interprocessor communication.

If this multiplicity of control units is combined with a new data communication strategy, a novel multiport memory architecture and extensive use of hardware macros, a new powerful architecture will result.

This architecture promises to overcome the traditional limitations of data flow computers. The challenge of this new architecture is to increase the control throughput and at the same time to match the throughput of the control section to the throughput of the processing section. This implies:

- a) Hardware design
- b) Algorithm matching

We will be mainly concerned with hardware design and restrict ourselves only to comment on the algorithm matching.

First, however, a more comprehensive look at data flow architecture.

1.1 DATA FLOW COMPUTERS: Classical Architecture

The classical DFC architecture [26] (Fig. 1.04) consists of five sections (often given different names):

- 1. Activity store
- 2. Instruction queue
- 3. Processing elements
- 4. Update unit
- 5. Fetch unit

FIGURE 1.04 : THE CLASSICAL DATA FLOW ARCHITECTURE

THE BASIC MECHANISM: The fetch unit takes an instruction address from the instruction queue. It then reads the activity template, pointed to by the instruction address, from the activity store. The fetch unit forms the template into an operation packet and passes it to the processing element. The processing element then performs the operation specified by the template opcode on data contained in the operation packet.

At the conclusion of the computation the processing element generates one result packet for each destination field of the operation packet.

THE ACTIVITY STORE holds the data flow program describing the computation to be performed as a collection of activity templates (Fig. 1.05 and 1.06). Each template contains the operation code, space for operands and a destination address for the result of computation. The destination could be another template if the computation is to be continued, or I/O if the computation process is completed.

THE INSTRUCTION QUEUE holds the addresses of individual activity templates which have complete operand fields.

PROCESSING ELEMENTS are the actual production tools of the computational process. The processing elements receive the activity templates, perform the appropriate computation, and send the result to the destination address specified by template. Common examples of functions available in processing elements are addition, subtraction,

FIGURE 1.05 : AN ACTIVITY TEMPLATE

a) $Z = (X + Y) \times (X - Y)$

FIGURE 1.06: INTERCONNECTION OF OPERATORS

multiplication and division. Since many processing elements could be employed in parallel, computations such as matrix multiplication and addition can be greatly speeded up in comparison with the essentially sequential processing of a conventional von Neuman machine.

THE UPDATE UNIT receives result packets and enters the values they carry into operand fields of activity templates as specified by their destination fields. The update unit also tests whether all operands and acknowledge packets required to activate the destination instruction have been received. If all the tests prove positive, the instruction (template) address is entered in the instruction queue for processing.

THE FETCH UNIT takes an instruction (template) address from the instruction queue and reads the activity template from the activity store, forms it into an operation packet and transmits it to the processing elements.

The instruction queue will be empty at the conclusion of the computation. A data flow computer with an empty instruction queue will await the next program ready for execution. A new program typically comes from an I/O device such as operator's terminal, host computer, etc.

A point to note here is that the execution of the program is initiated by the flow of data. In contrast to the above, a von Neuman architecture has its execution typically initiated by flow of instructions.

ADVANTAGES OF THE CLASSICAL DATA FLOW ARCHITECTURE

1. Implicit, automatic synchronization of many processors without a disadvantage of a complex operating system necessary for scheduling processes and processors in a conventional multiprocessor architecture.
2. A high degree of parallelism can be achieved for matrix operations and digital signal processing [3, 19, 20, 21]. This can be achieved without the usual penalty of high overhead [10] for process switching and synchronization.
3. Use of inexpensive VLSI building blocks [13] for implementation of processing elements and on packet switching functions. This usually implies a better cost/performance ratio than single processor or multi-processor units.
4. Easier programming (lower cost) for inherently concurrent (parallel) algorithms such as matrix manipulation, digital signal processing, weather forecasting, air traffic control, seismic studies (earthquake predictions, oil, gas and mineral explorations), finite element analysis for civil and mechanical engineering, etc.

1.2 LIMITATIONS OF DFC

Despite the many advantages data flow computers offer, they have not become a common computational engine for applications requiring a high degree of parallel processing. The list below shows some reasons why.

- a) Bus transfer rate severely limits the performance as all the data has to pass through the bus several times during the computation.
- b) The number of processors that can be attached to the bus is limited by the bus transfer rate. With today's technology, the number of processors that can be attached to the bus is so small that it precludes the construction of even a modest array of processors. This is especially true if the individual computational steps require transmission of large amounts of data as in FFT, matrix manipulations, etc.
- c) The requirement to acknowledge each packet creates a large computational and communication overhead.
- d) The interprocessor communication is achieved only with great difficulties as the data has to pass through the bus and all the control logic before interprocessor communication can be achieved.
- e) Each piece of data has to pass through the update and fetch units for each computational step. The data has to be

continually disassembled and reassembled into different templates. This creates a large computational overhead which tends to slow down the overall program execution speed.

- f) The programming is done in small steps instead of large macros. This tends to put an additional burden on the programmer and therefore decreases the efficiency of software production, making the machine difficult to use. As the programs consist of a large number of very small steps, the programs tend to be very tedious to write and long to execute. Use of large macros in the form of hardware chips would greatly increase the programming efficiency and parallelism of program execution.
- g) Lack of a suitable software language specifically designed for DFC makes it very difficult to make use of the inherent parallelism of the machine. Because of this limitation, problems that would lend themselves naturally to execution on DFC computers have to be laboriously adapted for execution on these machines.

It would be very useful to be able to predict numerically the efficiency of a computer architecture. We could then compare different computer architectures for efficiency, or perhaps design a novel computer architecture with a maximum efficiency for a given task.

PRESENT DATA FLOW MACHINES [39]

The technical literature describes the achievements in data flow computers to date. Among them are:

1. The Texas Instruments experimental data flow computer [29].
2. The MIT data flow supercomputer [26].
3. The University of Manchester (England) data flow computer [25].

The above architectures are shown in FIG. 1.07. We will now compare these architectures to the one shown in FIG. 3.2.2 in the context of maximizing their architectural efficiency and throughput.

The Texas Instruments DFC

1. Its single bus, called a ring, may become a bottleneck in a system with many processors. A single point failure in the bus may mean loss of a significant portion of computational resources.
2. Only partial distribution of control. Task allocation for individual processors is performed in a centralized location (TI 990/10). Front end interface may get congested in a

Texas Instruments Distributed Data Processor

M.I.T. Data-Flow Computer

Manchester Data-Flow Computer

FIGURE 1.07 : COMPARISON OF DATA FLOW ARCHITECTURES

system with many processors.

3. Interprocessor communication is achieved with difficulty.

The MIT Cell Block DFC

1. Non-modular structure grouped into blocks.
2. Very complex and potentially costly interconnection network between processors. A system with many processors may have a congestion problem.
3. Only partial distribution of control. Division of tasks among processors has to be performed by an additional machine or the programmer.
4. Slow communications between groups of processors.

University of Manchester DFC

1. Single main bus structure may become a bottleneck in a system with many processors. The bus width is unusually large - 96 bits.
2. Partially centralized control, operated by an LSI-11 computer. This centralized portion contains the template store and instruction queue.

The following proposal suggests a thesis which attempts to improve the data flow architecture by looking at its architectural efficiency.

1.3 THE THESIS PROPOSAL AND SUMMARY OF RESULTS

It is proposed that a high performance data flow computer must be designed so as to achieve a high architectural efficiency. This involves a detailed representation of overheads and the determination of a mechanism for reducing them to a minimum. The major overhead item, data movement, should be examined in detail. The control of such a distributed architecture is also a subject for detailed examination.

In order to accomplish this, the following steps are proposed:

1. Analyze the requirements for a data flow computer used for high performance digital signal processing.
2. Suggest several ways of improving the efficiency and performance of present data flow architectures.
3. Uncover the variables that govern the efficiency of a computer architecture.
4. If possible, derive numerical relationships between the architecture and its processing efficiency.

5. Develop a novel high performance data flow architecture.
6. Outline the performance possibilities of the novel DFC.
7. Outline further research needed in order to realize the suggested novel high performance data flow architecture.

SUMMARY OF RESULTS

Chapter one contains a critique of present data flow architectures. Chapter two derives a model for a data flow architecture and derives relationships for numerical calculation of architectural efficiency. Several ways of performance improvement are examined in this chapter.

Chapter three contains a design proposal for a novel high performance data flow architecture, as well as analysis of its performance.

Chapter four discusses the results.

The major results from this thesis are:

1. Use of the concept of architectural efficiency.
2. A detailed model for analysis of architectural efficiency of data flow computers.
3. An implementation model for data transfer by address which substantially improves the efficiency of the DFC.

4. Solution of the bus congestion problem by the use of multiport memories. Section 3.2 gives a more detailed description of this type of memory.
5. Use of VLSI macros to decrease bus traffic and control overhead.

2.1

2. ISSUES IN THE DESIGN OF A DATA FLOW COMPUTER

2.0 INTRODUCTION

This chapter presents the tools for improving the architecture of data flow computers. The topics discussed in this chapter are:

- a) Architectural efficiency measures
- b) System issues
- c) Implementation issues

The architectural efficiency measure presented in this chapter is a general purpose yardstick used to determine the efficiency of a computational engine. Thus, different designs can be compared and the most efficient computational engine can be selected.

The chapter is concluded by a summary of key concepts.

2.1 ARCHITECTURAL EFFICIENCY MEASURES

Section 1.0 presented the general expression for efficiency of a computer architecture as:

$$A.E. = \frac{E_u}{E_u + E_c}$$

In the case of a data flow computer architecture, the architectural efficiency can be calculated from the computational effort required for each operation within the circular pipeline.

Seven separate computation efforts can be identified within the DFC architecture:

- E_{FD} ... packet firing and distribution
- E_{RD} ... packet acknowledge and decoding
- E_{PR} ... processing
- E_{AR} ... packet acknowledge and result packet generation.
- E_{CD} ... collection and decoding of packets
- E_{AC} ... new packet assembly and check for completeness.
- E_{PE} ... control overhead in the processing element.

From the above computational efforts, only the processing (E_{PR}) constitutes useful output. All others constitute overhead. The whole situation can be shown graphically in a very convenient way, using flow graphs, (FIG. 2.1.1).

From the above, we can derive the expression for the architectural efficiency of a data flow computer:

$$A.E. = \frac{\sum_{K=1}^N E_{PR}^K}{E_{FD} + E_{CD} + E_{AC} + \sum_{K=1}^N E_{RD}^K + E_{PR}^K + E_{AR}^K + E_{PE}^K} \quad (2.1)$$

FIGURE 2.1.1 : MODEL FOR THE DATA FLOW COMPUTER ARCHITECTURE

DISCUSSION

In order to achieve a high architectural efficiency, all overheads will have to be reduced relative to E_{PR} . The reduction of overheads can be accomplished by reducing the number and the size of packets. Two tools can be employed towards this end:

a) Communication by address.

This technique reduces the size of packets by transmitting the address of data rather than the data itself.

b) Utilization of high level macros

This technique decreases the number of packets needed for a given computation. The macro performs a high level algorithm such as FFT or row by column multiplication with an insignificant amount of overhead. This significantly decreases the number of packets transmitted over the buses.

Both of these techniques are discussed in more detail later in this chapter.

The following examples will illustrate the above concept.

Example 2.1 DFC with 1 x 1 word multiplier

Let a data flow computer with five sixteen bit multiplier units have the following parameters:

$$E_{FD} = 3 \text{ MIPS}$$

$$E_{RD}^1 = E_{RD}^2 = \dots = E_{RD}^5 = 5 \text{ MIPS}$$

$$E_{PR}^1 = E_{PR}^2 = \dots = E_{PR}^5 = 10 \text{ MIPS}$$

$$E_{AR}^1 = E_{AR}^2 = \dots = E_{AR}^5 = 8 \text{ MIPS}$$

$$E_{CD} = 9 \text{ MIPS}$$

$$E_{AC} = 20 \text{ MIPS}$$

$$E_{PE}^1 = E_{PE}^2 = \dots = E_{PE}^5 = 1 \text{ MIPS}$$

The architectural efficiency of the above DFC can be calculated from equation 2.1:

$$A.E. = \frac{\sum_{K=1}^5 E_{PR}^K}{E_{FD} + E_{CD} + E_{AC} + \sum_{K=1}^5 E_{RD}^K + E_{PR}^K + E_{AR}^K + E_{PE}^K}$$

$$A.E. = 0.33 = 33\%$$

Example 2.2 DFC with 10 x 10 array multipliers

Let a data flow computer with five 10 x 10 sixteen bit array multipliers have the overheads reduced by a factor of five as is compared to example 2.1. Let the above computational engine have the following parameters:

$$E_{FD} = 0.6 \text{ MIPS}$$

$$E_{RD}^1 = E_{RD}^2 = \dots = E_{RD}^5 = 1 \text{ MIPS}$$

$$E_{PR}^1 = E_{PR}^2 = \dots = E_{PR}^5 = 10 \text{ MIPS}$$

$$E_{AR}^1 = E_{AR}^2 = \dots = E_{AR}^5 = 1.6 \text{ MIPS}$$

$$E_{CD} = 1.8 \text{ MIPS}$$

$$E_{AC} = 4 \text{ MIPS}$$

$$E_{PE}^1 = E_{PE}^2 = \dots = E_{PE}^5 = 0.2 \text{ MIPS}$$

The architectural efficiency of the above DFC as calculated from equation 2.1 will equal:

$$A.E. = 0.71 = 71\%$$

2.2 SYSTEM ISSUES

From a systems viewpoint, the main issues are:

- 1) Efficiency.
- 2) Throughput.
- 3) Configurability for different tasks.

- 4) Location of control.
- 6) Data communication strategy.
- 7) Bus architecture.
- 8) Graceful degradation of performance in the event of a failure in some part of the computational engine.
- 9) The ability to expand the architecture in discrete steps to create computational engines with a wide variety of throughputs.
- 10) Others.

The above requirements clearly lead to a computational engine with a high degree of parallelism. An architectural configuration with many identical computational elements will satisfy the requirements for throughput and graceful degradation of performance. Reliability can be achieved by a high level of integration, i.e., VLSI chips. The main issues that remain to be solved, are the ones of efficient architecture, location of control, data communication strategy, and bus structures. A closer examination will reveal that the latter four issues are actually inter-related and the proper choice for each of them will lead to a more efficient architecture with a substantially increased throughput.

We will now examine each of these issues in greater detail.

LOCATION OF CONTROL

Two broad classes can be identified with respect to the above issue:

- a) Centralized control

b) Distributed control

The location of control is closely related to the throughput capabilities of the control unit. In a centralized control unit, all the control traffic has to pass through one central unit. This unit may get overloaded when a large number of processors is used.

The situation is much improved when the control is decentralized. The multiplicity of control units in a decentralized control unit contributes to a significant increase in the throughput of the control unit by sharing the processing among many smaller control units which operate concurrently.

CENTRALIZED CONTROL (Fig. 2.2.1)

By centralized control we mean that all the decisions on allocation of computation resources, queuing, branching, packet assembly, error checking, etc, are performed in a central location.

The processing elements are not involved in any decision at all; they simply perform the computation as directed by the centrally located control unit. Therefore the computational elements themselves are relatively simple logical structures. The complexity of the machine is concentrated in one central location.

The Control Throughput of a centralized control unit equals:

$$C.T. = E_{FD} + E_{RD} + E_{AR} + E_{CD} + E_{AC} + \sum_{K=1}^N E_{PE}^K \quad (2.2)$$

N ... number of processors

FIGURE 2.2.1: CENTRALIZED CONTROL

DISTRIBUTED CONTROL

By distributed control we mean that some or all of the decisions on allocation of computational resources, branching, queuing, packet assembly, error checking, etc., are distributed throughout the system.

FULLY DISTRIBUTED CONTROL (Fig. 2.2.2)

CONCEPT: This system is characterized by complete distribution of the control function in the machine.

IMPLEMENTATION: One possible implementation of such a machine is a dual bus system. The processors communicate with each other through two bidirectional buses (Bus 1, Bus 2). All the queuing, packet assembly, error checking, etc., is accomplished by the processors. All the information in the system is available to all processors. Inter-processor communication is accomplished through multi-port memories.

IMPLICATIONS:

- 1) Each process has its own queue.
- 2) As there is free communication between the memories of all processors, all the process information is easily shared by all the processors.
- 3) Data, templates and results can be freely exchanged between the processors.
- 4) This configuration tends to be memory intensive.

FIGURE 2.2.2 : FULLY DISTRIBUTED CONTROL

**THE CONTROL THROUGHPUT OF A FULLY DISTRIBUTED
CONTROL UNIT EQUALS:**

$$C.T. = \sum_{K=1}^N E_{FD}^K + E_{RD}^K + E_{AR}^K + E_{CD}^K + E_{AC}^K + E_{PE}^K \quad (2.3)$$

N ... number of processors.

Distribution of control has the following advantages:

- 1) Significantly increased control throughput due to concurrent operation of a large number of control units.
- 2) Increased modularity of the architecture.
- 3) Graceful degradation of performance in the event of a failure in one of the control units. This implies higher reliability compared to a single controller architecture.
- 4) Reduced traffic on the main bus. A large portion of the packet traffic is performed over the local buses.

DATA COMMUNICATION STRATEGY

By data communication strategy we mean the method used for communicating data between the different parts of a computational engine.

Three broad strategies can be identified:

- a) Communication by value
- b) Communication by address
- c) Other

COMMUNICATION BY VALUE: Communication by value is the most common method used today. By using this method, all the data required in the system has to be physically transmitted over the circular pipeline, connecting all parts of the computational engine. This, of course, creates a large volume of traffic in the pipeline. Therefore, this strategy is only useful for computational engines of low performance. In a high performance machine the large volume of traffic in the pipeline would tend to make it the bottleneck of the system. This will become ever more apparent when communicating with VLSI chips such as FFT transformers, which require a large number of data points. High performance machines will demand a new solution to this problem.

COMMUNICATION BY ADDRESS: This data communication strategy is based on the premise that all the data is available to all processing units at all times. To access a data block, only the starting address and the length of the block is transmitted.

This method is very memory intensive and requires a different memory architecture from the memory systems currently in use.

However, keeping in mind that semiconductor memory size and cost are improving rapidly, we see this method as very promising for high performance systems. A memory system suitable for this application will be described in Chapter Three. What makes this method so attractive is the substantial reduction of data rate over the common bus. The large, more complex high performance chips in today's machines demand ever increasing amounts of data to be transmitted to each chip. In addition to

that, the trend in today's computer architectures is toward a higher degree of parallelism and therefore a demand for buses, transferring large amounts of data.

By using this method of communication by address, the amount of information transmitted over the buses is greatly reduced. This improves the efficiency of the computational engines by decreasing overhead generated by bus transfers. The advantages of the above concept are best illustrated by an example:

EXAMPLE 2.3

If we assume a 1024 point FFT, communicating by value would require 1026 words (Fig. 2.2.3 a) For the same 1024 point FFT, communicating by address would require four words (Fig. 2.2.3.b). The resultant reduction in data bus transfer rate would amount to:

$$K = \frac{N_1}{N_2} = \frac{1026}{4} = 256 \text{ times}$$

The data bus transfer rate would decrease by a factor of 256, virtually eliminating bus congestion — a problem common with classical DFC architectures.

If we consider a 10,000 point FFT, the figures become even more appealing:

Improvement factor for the data transfer rate.

$$K = \frac{N_3}{N_2} = \frac{10,002}{4} = 2,500 \text{ times}$$

a) PACKET FOR COMMUNICATION BY VALUE

b) PACKET FOR COMMUNICATION BY ADDRESS

FIGURE 2.2.3 : DATA PACKETS FOR EXAMPLE 2.3

BUS ARCHITECTURE AND MULTIPOINT MEMORIES

The bus system is an electronic highway used for delivery of information from one section of the computational engine to another. As such, it should be as speedy and error free as possible. The handling of error detection and correction is normally achieved by special chips for error detection and correction.

The transmission speed of the buses can be increased to a certain extent by the usage of more advanced technology. However, the best solution to the bus congestion problem would be a multipoint memory system with several interconnecting buses.

The multipoint memories would automatically interleave the information from the buses, so that the processors would seem to see a single bus with a very high bandwidth.

This multipoint memory system is more fully described in Section 3.2.

2.3 IMPLEMENTATION ISSUES

Some common implementation issues are:

- 1) Environmental factors (space, military, commercial).
- 2) Technology transparency.
- 3) System partitioning for implementation.
- 4) Implementation of control algorithms (hardware vs. software)

trade-off).

- 5) Modularity, use of common or special parts.
- 6) Cost and price/performance ratio.
- 7) Ease of repair.
- 8) Life cycle cost.
- 9) Ease of production (i.e. does the architecture lend itself to mass production).
- 10) Others.

ENVIRONMENTAL FACTORS: Environmental consideration influence the packaging, size, power dissipation and cost among others. Space and military applications demand small size, reliability and ruggedness, but do not place rigid restrictions on cost. Commercial applications usually are more price sensitive, but tend to put less emphasis on size and ruggedness.

TECHNOLOGY TRANSPARENCY is a very important consideration. A good architecture should be implementable in at least several semiconductor technologies to allow for improvement in performance and upgrading as better technology becomes available. An example of an architecture that has been implemented in several technologies is the PDP-11.

SYSTEM PARTITIONING: Proper system partitioning minimizes the number of different subsystems that have to be manufactured, thereby decreasing cost and easing repair when it becomes necessary.

IMPLEMENTATION OF CONTROL ALGORITHMS: Hardware implementations of control algorithms tend to run faster and for large production runs decrease the cost of implementation.

MODULARITY: Modularity and use of commonly available parts decreases cost and eases maintenance during the life cycle of a machine. If an architecture requires special parts that are difficult to manufacture, the whole implementation becomes very costly and sometimes impractical. Examples of machines with many special parts are IBM 360, CRAY-1 etc.

COST AND PRICE/PERFORMANCE RATIO: Cost and price/performance ratio can be improved by using VLSI implementations and as many standard parts as possible. Highly parallel architectures tend to use a large number of identical parts, which decreases the cost of manufacturing.

LIFE CYCLE COST: This includes ease of production, ease of repair by incorporating standard modules, diagnostic hardware and software. Easy software maintenance implies modular software and high reliability of the software and hardware.

2.4 **THE MYTH ABOUT UNLIMITED EXPANDABILITY OF DFCs**

There is a widespread myth in the computer industry that basic data flow computer architectures can be limitlessly expanded in all directions. Unfortunately, for the sake of progress, this myth has to be shattered. As a matter of fact, classical data flow architectures suffer

from at least two serious flaws that limit their expansion potential:

- 1) Bus congestion
- 2) Centralization of control in a pipeline with a limited number of stages

One of the main problems of classical data flow computer architectures is the centralization of control. No matter how fast the control pipeline will work, there will come a point where the number of processors is just too large to be handled by a centralized pipeline.

What is needed is a fully distributed control section which has a multiplicity and performance that is matched to the performance of the processing units. When both the control unit and the processing units are distributed and modular, the performance of the computational engine can be incrementally expanded to match the varying requirements of different applications.

Data bus congestion is also a serious limiting factor for classical data flow architectures. The large number of packets, transmitted over a single data bus, limits the number of processing units that can be attached to the bus. The data bus congestion problem can be eliminated by using a multiport memory with several buses. The multiport memory is described in Section 3.2.

2.5 SUMMARY OF DESIGN ISSUES

This chapter presented the tools for improving the efficiency and throughput of computational engines. The main issues presented here were:

- 1) The architectural efficiency measure — a tool for establishing the efficiency of a computer architecture.
- 2) Decentralization of control eliminates the bottleneck in the control section.
- 3) Data communication by address decreases data bus traffic to streamline the transfer of data between the various parts of the computational engine.
- 4) Hardware macros decrease control overhead.
- 5) Technology transparency for easy upgrading with technological advances.
- 6) Modularity for easy expandability.
- 7) Ease of repair.
- 8) Life cycle cost.
- 9) Ease of production for low initial cost and mass production.

The next chapter will use the above tools for design of a novel high performance data flow architecture for digital signal processing.

3.1

3.0 A NOVEL HIGH PERFORMANCE
DATA FLOW ARCHITECTURE

3.0 INTRODUCTION

This chapter uses the tools developed in chapter two to accomplish a design of a novel high performance data flow architecture for digital signal processing. This chapter will first analyze the operational requirements for digital signal processing, then present the new architecture, compute its performance characteristics and conclude with a summary.

3.1 OPERATIONAL REQUIREMENTS FOR DIGITAL SIGNAL PROCESSING

Operational requirements for DSP are fourfold in nature: Computational Requirements, Throughput Requirements, System Control and Synchronization, Architectural Flexibility.

These will be considered in the following section.

3.1.1 COMPUTATIONAL REQUIREMENTS

Digital signal processing is used in speech recognition, speech synthesis, filtering, sonar and radar filtering, radar and sonar target identification and recognition, Kalman filtering, seismic signal processing and others.

After carefully analyzing the computational requirements for each of these applications, we come to realize the commonality of the computational techniques used in each application. It turns out that only

seven types of algorithms are sufficient to handle any digital signal processing problem. We will call these basic algorithms BASIS FUNCTIONS [8]. Most digital signal processing problems involve one or several of the basis functions at the time. Only on very rare occasions all seven basis functions are used to solve one problem. The basis functions involve the following algorithms:

1. Difference Equation Calculation

$$Y(n) = \sum_{k=0}^N a_k X(n-k) - \sum_{k=1}^M b_k Y(n-k) \quad (3.1)$$

This equation is used for a recursive or infinite impulse response filtering operation. Here $X(n)$ is the input sequence and $Y(n)$ is the filtered output sequence. The parameters N , M , a_k and b_k define the transfer function of the filter.

If all the coefficients b_k are zero, then the above equation reduces to the form of a finite convolution sum, representing a finite impulse response digital filtering operation.

2. Correlation Coefficient Calculation

$$\Phi(n) = \sum_{k=-N}^N X(n+k) Y^*(k) \quad (3.2)$$

In this equation, the interval $-N$ to N represents the window over which

the two signals are correlated and $Y^*(k)$ represents the complex conjugate of $y(k)$. The two signals to be correlated are $X(k)$ and $Y(k)$.

3. Complex Frequency Translation

The translation of a digital signal to a different frequency can be represented as a complex multiplication:

$$y(n) = X(n)e^{j2\pi nk} \quad (3.3)$$

The parameter k represents the ratio of the translation frequency to the sample frequency of the input sequence $X(n)$.

4. Discrete Fourier Transform

This is one of the most widely used calculations in DSP. The computation of discrete Fourier transform coefficients can be represented by the equation:

$$X(k) = \sum_{n=0}^{N-1} X(n)e^{-j(\frac{2\pi}{N})nk} \quad (3.4)$$

and the inverse Fourier transform:

$$X(n) = \frac{1}{N} \sum_{k=0}^{N-1} X(k)e^{-j(\frac{2\pi}{N})nk} \quad (3.5)$$

5. Power Spectral Density or Magnitude Squared

This is another commonly used calculation. It can be represented by the equation:

$$[X(k)]^2 = X(k)X^*(k) \quad (3.6)$$

6. Matrix Arithmetic and Matrix Manipulations

The fact that many signal processing problems can be easily formulated in vector/matrix format leads to the requirement for matrix operations. The arithmetic requirements are addition, subtraction, multiplication of variable dimension matrices, scalar and cross vector multiplication. The most commonly used matrix manipulations are inversion, transposition and complex conjugation.

7. Logarithms and Exponentiation

These operations should be available to base two, base ten and any other base possibly required for special applications.

CONCLUSIONS

The above computational requirements imply algebraic operations on complex numbers with the associated increase in computational complexity and volume.

A requirement for calculation of a large number of sums of products also exists.

In some cases (as in FFT), the data has to be presented to the computational units in a non-sequential manner prescribed by the algorithm. This necessitates a highly sophisticated and efficient system of "next" address calculation unit.

One of the key requirements for digital signal processing is the ability to handle large volumes of computations, such as sums of product, per unit of time. This implies efficient data handling, and address calculations as well as flexibility in architecture in order to adapt to different classes of problems.

3.1.2 THROUGHPUT REQUIREMENTS

REPRESENTATION

Each signal sample can be represented in the digital domain as a fixed point or floating point digital number. The digital word length in each case can be from 8 bits to 64 bits. The number of bits selected depends to a large extent on the required signal to noise ratio and dynamic range of the signal. As the signal to noise ratio is:

$$S/N = 6N \text{ (dB)}$$

where N = number of bits, we can see that for a signal to noise ratio of 96 dB a word length of 16 bits would be required. Likewise, eight bit

representation would yield a signal to noise ratio of 48 dB which is rather poor for most applications. On the other hand, a long data word means larger buses or slowdown in data movement because of the requirement of multibyte transfers for each data value.

VOLUME

A typical digital signal processing application requires many data points. In audio applications (speech processing, music processing, special effects, etc.), the maximum frequency of interest would be 25,000 HZ. To satisfy Nyquist criterion, we must produce at least 50,000 samples per second, each of which would be a sixteen bit word. In practice, however, we would probably produce 75,000 samples per second to ease the requirements for post-filtering.

In video (television, VTR, special effects, etc.), the maximum frequency of interest would be five megahertz for present systems. This may become higher with the switch-over to high quality television with 1000 lines resolution. A five megahertz system may require processing of ten million samples for each second of transmission.

Radar and multisensor sonar applications demand even higher volumes of data throughput. In radar, a typical bandwidth can easily reach twenty megahertz, necessitating the processing of forty to sixty million samples per second. In radar applications, we have to remember that the spatial resolution of the equipment is directly proportional to the frequency being used. As the radar technology matures, one can expect

a switch to a higher bandwidth with the resultant requirements for higher speed and higher volume processing.

TIME

Processing time is of equal importance for many applications. In every application, all processing must be completed between two successive samples or sample sets (as in FFT) in order to achieve real time response.

Consider a high speed transmission channel, operating at ten megahertz. For echo cancellations, we would require two FFTs, two inverse FFTs and one digital filtering operation. We assume a 1024 point FFT and the same time required for each of the above operations.

We would then process between twenty and thirty million operations per second. The time available for each operation would then be approximately between six and ten microseconds for an 1024 point FFT.

For audio applications, with 75,000 samples per second, the time available for each of the above operations would be approximately three milliseconds.

However, many applications require far more complex processing. The time constraint for such applications is obviously much more severe than in the examples mentioned.

In summary, we can see that the throughput requirements for a given application necessitate an architecture with a high degree of parallelism and efficient mechanism for high volume data movement.

3.1.3 SYSTEM CONTROL AND SYNCHRONIZATION

In a typical data flow computer, many processes can be run concurrently due to the inherent parallelism of the architecture. Obviously, with many processes running concurrently, there is a need for process synchronization at a high level to avoid confusion. In order to complete a certain task, many processes may have to be run in a particular sequence. In some cases use of the resources may require cooperation in fulfilling the overall task. Under some operations it is necessary to insure that the results of various operations leave the resource in a determinant state (logical or logically consistent state). Also, a scheduling mechanism must insure that the a deadlock state does not occur. That means the scheduling or exclusion mechanism must not under any particular set of circumstances prevent any use of the resources.

All the above synchronizing mechanisms imply that we must devote part of the raw computational power to synchronization and control. This detracts the computer system from its main objective — producing useful work. The necessary overhead associated with synchronization and control functions can become quite large as the number of processors and processes increase. An additional burden is

the fact that somebody has to write, debug and maintain the highly intricate software required in the operating system.

In large systems the synchronization overhead associated with the addition of another processor may be so large that the computational benefits would turn out to be only marginal.

In data flow computers the situation is eased due to automatic synchronization among different processing units.

Depending on the requirements of each particular algorithm, a high degree of parallelism or a strict precedence may exist in the computation. For an algorithm with a high degree of parallelism, the data flow computer offers a significant speed improvement of processing. An algorithm that requires a strict precedence relationship with sequential processing will run at the same speed as it would on the von Neuman architecture.

3.1.4 ARCHITECTURAL FLEXIBILITY

One basic requirement is architectural flexibility and dynamic reconfigurability under software control. This requirement has to be accommodated in order to satisfy changing requirements for different applications. The basic architecture has to be able to adapt to varying requirements of different users.

The scale of requirements can range from one of strict procedural precedence to a fully parallel configuration.

For digital signal processing a fully parallel configuration would probably be preferred because of the inherent parallelism in the algorithm. Other applications, such as spacecraft control and others, would use a more sequential algorithm. Data bases and data acquisition applications tend to be spatially distributed. Their architectural requirements would be different from a localized application such as a matrix processor in a supercomputer.

Ideally, all the above requirements should be accommodated in an architecture that would have both performance and flexibility at the lowest cost. An approach that promises to fulfill just this requirement will be described in the next section. The architecture makes extensive use of standard inexpensive VLSI chips that promise high performance at very low cost. This is possible because the individual modules of this architecture lend themselves to mass production. Its computational power can be expanded in incremental steps, therefore accommodating a wide variety of requirements.

3.2 A NOVEL HIGH PERFORMANCE DATA FLOW ARCHITECTURE

The novel architecture described below incorporates all the main features described in chapter two:

- 1) Decentralized control
- 2) Data communication by address
- 3) A multibus system, capable of broadcasting data to many processing units simultaneously.
- 4) Hardware macros.

Two variations of the architecture have been considered:

- 1) Segmented memory architecture Fig. 3.2.1
- 2) Multiport memory architecture Fig. 3.2.2

In the segmented memory architecture (Fig. 3.2.1), each processor communicates with one memory segment of each processor, i.e. processor #1 communicates with memory segments 1 of all processors, processor #2 communicates with segment 2 of all processors, etc.

This has a disadvantage of each processor having access to a limited portion of the memories of the other processors. The information is broadcast to many memory segments simultaneously.

FIGURE 3.2.1 : SEGMENTED MEMORY ARCHITECTURE

In the multiport memory architecture (Fig. 3.2.2), each processor can communicate to any memory location by broadcasting the address and data over one of the buses. If necessary, the communication could be made processor selective by adding additional bits to address. The number of buses is from one to any number, as required by the application. For each additional bus, each of the memory modules $M(1)$ to $M(N)$ require an additional port for data and address.

Both architectures (Fig. 3.2.1 and 3.2.2) require a port sequencer in each memory module. Apart from the above differences, both architectures consist of the same basic building blocks. The operation of both architectures is identical. The novel architecture consists of N identical sections, each of which is composed of:

- 1) The processing element PE (i.e. multiplier, adder, FFT transformer, matrix multiplier, etc.) The emphasis is on functional units that perform large "macros".
- 2) Control units consist of instruction queue, template units, update units, and fetch units.
- 3) Memory section
- 4) Interconnecting buses.

FIGURE 3.2.2 : MULTIPORT MEMORY ARCHITECTURE

FIGURE 3.2.3: THE MULTIPOINT MEMORY

During the program execution, the instruction queue of each control section C_1 to C_N holds all the instructions necessary for completion of the whole algorithm. The memory section of each processing element holds all the data necessary for the whole algorithm. The intermediate results as well as acknowledge packets are broadcast to all processing elements during execution as required.

The control unit of each section contains templates appropriate for the processing element used in that section.

THE MULTIPORT MEMORY (Fig. 3.2.3)

The high performance data flow architectures of Figures 3.2.1 and 3.2.2 rely on high speed multiport memories for their operation.

As the memories constitute an important part of the overall system, we will examine them in greater detail.

Multiport memories consist of three major parts:

1. Bidirectional packet ports.
2. Controller.
3. Memory array.

Packet Ports

Each packet port is connected to one of the buses B_1 through B_k as shown in Fig. 3.2.2. A port consists of a bidirectional bus driver with

a word width appropriate for the bus to which it is connected and a BUSY line that can be tested by the controller.

The packet ports are normally in the RECEIVE mode, unless the controller is using them for transmission of a packet to other processors.

The Controller

The controller is continuously scanning all the packet ports in order to determine if a new packet is being sent over the bus.

The arrival of a new packet is usually preceded by an active BUSY line. An active BUSY line also tells the controller that the port cannot be used for transmission of an output packet.

The controller must have enough intelligence to interpret the appropriate destination for the packet and direct its contents to the proper place in all memory arrays. The destination address is usually contained in the packet itself. By using fast ECL circuits, the address transfer from the packet to memory should take only a few nanoseconds. The address transfer time could be further reduced in the future by using more advanced address circuitry.

The controller acts as a concentrator of traffic from the packet ports into the memory.

The much higher traffic density in the memory array does not have

to be a problem in a properly designed system. Memories with six nanosecond access time at a four kilobit density level are available today. [27, 28]. The near future will bring memories of much higher densities and speeds. Work on one nanosecond GaAs RAMs has been announced recently by a company called NANOTEC.

The communication between the memory and the processing element and its control section is accomplished through a set of buffers. One section of the buffers serves the bidirectional data bus, the other section serves the address bus. The MEM REQ and READY lines are used for synchronization of data transfers between the memory and the processing element and its control units.

In some cases, the processing element and its control unit could be merged into one unit.

The Memory Array

The memory array has a conventional configuration with separate address and data buses. It is divided into many smaller memory banks which are activated only one at a time. This arrangement enables us to activate only a small portion of the memory each time, while the rest of the memory array is in a power conserving deselected state. As a result, the memory array consumes only minimal power.

By emphasizing a multiport memory, we took the congestion problem from the buses to the memory because fast memories are easier to

get than fast buses and more likely to appear.

Long distance communication over the buses is therefore relatively slow, while short distance communication inside the memory array can be quite fast.

In order to give the reader some feel for the memory technology that will be required, we will present three cases.

Case 1

Number of buses being served is $T = 2$, bus bandwidth is $B = 5$ MHz each, the packet address recognition time is given as $ART = 2$ nanoseconds, and BUSY line recognition time $BUSY = 2$ nsec. Calculate the minimum memory access time required.

Solution:

$$MAT = \frac{1}{T \times B} - ART - BUSY$$

$$MAT = 96 \text{ nanoseconds}$$

Case 2

Number of buses being served is $T = 10$, bus bandwidth is $B = 5$ MHz each, the packet address recognition time is given as $ART = 0.5$ nanoseconds, and BUSY line recognition time $BUSY = 0.5$ nsec. Calculate the minimum memory access time required.

Solution:

$$MAT = \frac{1}{T \times B} - ART - BUSY$$

$$MAT = 19 \text{ nanoseconds}$$

Case 3

Number of buses being served is $T = 10$, bus bandwidth is $B = 50 \text{ MHz}$ each, the packet address recognition time is given as $ART = 0.1 \text{ nanoseconds}$, and $BUSY$ line recognition time $BUSY = 0.1 \text{ nanoseconds}$. Calculate the minimum memory access time required.

Solution:

$$MAT = \frac{1}{T \times B} - ART - BUSY$$

$$MAT = 1.8 \text{ nanoseconds}$$

THEORY OF OPERATION

The operation of this architecture can best be described by the following sequence of steps:

- 1) The instruction queue of each control section is filled by broadcasting the proper information to all sections simultaneously.
- 2) The memory sections are filled with the appropriate data by broadcasting it to all sections simultaneously.

- 3) As instruction template, corresponding to the first entry in the instruction queue is filled with addresses and transmitted to the appropriate processing element.
- 4) An acknowledge packet is broadcast to all other processors.
- 5) The other processors pick one of the next entries in the instruction queue as appropriate for their kind of processing element, observing any precedence requirement as specified in the instruction queue.
- 6) After finishing one instruction, the control unit of each section will pick the next appropriate instruction for its processing element, that has not been acknowledged.

Intermediate results are broadcast to all processors as required.

- 7) The END instruction in the instruction queue will stop the completed execution.

3.3 SUMMARY

Chapter three has accomplished the following:

- a) Analysis of operational requirements for digital signal processing
- b) Presentation of a novel high performance computer architecture for digital signal processing.

The novel architecture presented in this chapter is characterized by the following features:

- 1) Multiplicity of processing elements.
- 2) Decentralization of control
- 3) Data communication by address rather than by value
- 4) Broadcasting of data to many processing elements simultaneously.
- 5) Use of multiple buses for speedy inter-processor communication.
- 6) Use of multiport memory systems.
- 7) Self-synchronizing of many processors through use of acknowledge packets and distributed control.

4. CONCLUSIONS

4.0 INTRODUCTION

This chapter will deal with the following topics:

- 1) summary and conclusion
- 2) topics for further research

This section will present the research topics that have to be solved before this novel architecture can become a commercial success.

4.1 SUMMARY AND CONCLUSION

This thesis has introduced the concept of architectural efficiency.

This concept was then applied to data flow computers in order to attempt to improve their throughput and efficiency.

Several ways for improvement were suggested, among them the multiport memory and the use of hardware macros. The improvements resulted in suggestions for a novel architecture which reduces control overhead and bus congestion.

The extensive use of hardware macros, multiport memories and communication by address should decrease the control overhead and reduce the traffic over the main buses.

The novel architecture incorporates a high degree of parallelism and modularity. It also lends itself to mass production in the form of inexpensive VLSI components. The possibility exists that this architecture could become a cost effective vehicle for high performance digital signal processing.

4.2 TOPICS FOR FURTHER RESEARCH

This thesis has accomplished the following groundwork for the novel high performance data flow architecture:

- 1) It has established a measure for efficiencies of computer architectures.
- 2) A model for a data flow computer architecture has been established. Using this model, overheads have been identified.
- 3) A novel high performance data flow computer architecture has been devised.

In order to complete the development of this novel architecture, further research is required. This research falls into two broad categories:

- 1) Theoretical research
- 2) Implementation research/development.

1. Theoretical Research

In this category, two outstanding issues need exploration:

1. High level software languages
2. Reliability

One of the most important areas of theoretical research is the software generation for the novel architecture. This research should start with the selection of an optimal mix of templates for a certain algorithm, then proceed to a form of a macro assembly language and finally concentrate on the development of a suitable high level language. The availability of a user friendly high level language is crucial to the success of this novel architecture. Special care should be taken to make the language simple to use and easy to learn. The language should be applicable to a wide variety of problems.

Acknowledge algorithms and precedence coding is another facet of software, which warrants further research. Appropriate software debugging tools will be required for software development on the new machine.

Reliability of the novel architecture is very important. In order to maximize the reliability, research should be done in order to determine the vulnerability of the new architecture to failures of its individual parts and possible performance and reliability implications of such single or multiple point failures. Recovery mechanisms should be studied in case of bus failure or in the event that two or more processors send the

same acknowledge packet at the same time.

2. Implementation R/D

This section consists of the following outstanding issues:

1. Hardware/software implementation ratio.
2. Power distribution and thermal design.
3. Hardware implementation of the control section.
4. Optimum partitioning of the computational engine.

One of the challenges of the implementation stage is to find the best hardware/software implementation ratio for each application. Research should be done into methods that would enable the designer to determine the proper ratio analytically. The power distribution and thermal design of systems with a large number of processors will also present an interesting challenge. Research here should concentrate on the best way to distribute large amounts of power and on efficient ways of cooling a densely packed computational machine.

The control section used in every module of the computational engine, should be implemented in hardware. The hardware implementation would provide the maximum possible speed at the lowest cost. Research should be directed towards designing a suitable hardware architecture and then implement it perhaps as a gate array or a custom VLSI integrated circuit.

• The optimum partition of the whole computational engine into

suitable custom or standard integrated circuits or boards will certainly have to be researched in detail.

5.1

5. APPENDIX

5.1 APPLICATION AREAS FOR THE NOVEL DFC ARCHITECTURE

The potential for large throughput and high architectural efficiency can best be exploited in applications requiring a high degree of parallelism and when processing of large volumes of data is required in a short period of time.

Some applications of this nature are listed below.

- 1) Digital signal processing for sonar.
- 2) Weather forecasting.
- 3) Digital signal processing for radar.
- 4) DSP for signal recovery in communications (space and terrestrial).
- 5) DSP for telephony (S/N ratio improvement, voice data enhancements, bandwidth modification/translation and echo cancelling).
- 6) Control and supervision of a large number of telephone connections.
- 7) Air traffic control.
- 8) Process control of large plants.
- 9) Artificial intelligence and robotics.
- 10) DSP, signal improvements and modification for film production and audio recording companies.
- 11) Office of the future (message switching, processing of large amounts of data, etc.)
- 12) Image correlation.
- 13) Sophisticated video games, animation and flight simulators.

- 14) Military missile control.
- 15) Military command, control and communication
- 16) Military fire control.
- 17) Encryption and decryption of sophisticated codes. Code breaking.
- 18) Electronic counter measure and counter-counter measures.
- 19) Data base handlers for large data bases.
- 20) Intelligence in sophisticated satellites and space stations.
- 21) Data reduction in imaging systems.
- 22) Sophisticated voice recognition and speech synthesis.
- 23) Forensic sciences.
- 24) Data reduction in seismology, geophysics, astronomy, high energy physics.
- 25) Solution of large mathematical problems in real time.
- 26) Medical applications (diagnostics, treatment and image processing)
- 27) CAD/CAM and CIM
- 28) Large scale character recognition.

5.2 GLOSSARY OF TERMS

Von Neuman architecture

A computer architecture in which the execution process is initiated by instruction flow.

Data flow architecture

A computer architecture in which the program execution is initiated by flow of data.

Bus

Electronic highway, connecting all parts of the computer. It is used to transmit information between the different parts of the computer.

Memory

Section of the computer, used to store programs or data.

I/O

Stands for INPUT/OUTPUT. Means for communicating with humans or other machines.

5.3 REFERENCES

1. William B. Ackeyman, "Data Flow Languages", IEEE Computer, Feb. 1982, pp. 15-23.
2. Hassan M. Ahmed, Jean-Marc Delosme, and Martin Morf, "Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing". IEEE Computer, Jan. 1982, pp. 65-80.
3. Andreas Antoniou, "Digital Filters Analysis and Design". McGraw-Hill. (1979)
4. George H. Barnes and Stephen F. Lundstrom, "Design and Validation for a Connection Network for Many-Processor Multiprocessor Systems", IEEE Computer, Dec. 1982, pp. 32-41.
5. J.A. Bayliss et al, "The Interface Processor for the 32b Computer". ISSCC '81 Digest, pp. 116-117.
6. Robert Bernhard, "Giants in Small Packages", IEEE Spectrum, Feb. 1982, pp. 39-44.
7. J.W. Byers et al, "A 32b VLSI System". ISSCC '82 Digest, pp. 128-129.
8. B.A. Bowen and W.R. Brown, "Multiple Processor Architectures for Digital Signal Processing". Carleton University, Ottawa.
9. D.L. Budde et al, "The 32b Execution Unit". ISSCC '81 Digest, pp. 112-113.
10. R.J.A. Buhr et al, "Collected Technical Papers of the Microprocessor System Development Laboratory", Carleton University 1977.
11. Per-Erik Danielsson and Stefano Levialdi, "Computer Architectures for Pictorial Information Systems". IEEE Computer, Nov. 1982, pp. 53-67.
12. Alan L. Davis and Robert M. Kelleo, "Data Flow Program Graphs". IEEE Computer, Feb. 1982. pp. 26-41.
13. Douglas G. Fairbairn, "VLSI: A New Frontier for Systems Designers", IEEE Computer, Jan. 1982, pp. 87-96.
14. D.D. Gajski, D.A. Padue, "A Second Opinion on Data Flow Machines and Languages", IEEE Computer, Feb. 1982, pp. 58-68.

15. Arvind and Kim P. Gostelow, "The U-Interpreter". IEEE Computer, Feb. 1982, pp. 42-48.
16. Allan Gottlieb and J.T. Schwartz, "Networks and Algorithms for Very Large-Scale Parallel Computation". IEEE Computer, Jan. 1982, pp. 27-36.
17. Leonard S. Haynes, Richard L. Lau, "A Survey of Highly Parallel Computing". IEEE Computer, Jan. 1982, pp. 9-24.
18. W.W. Lattin et al, "A 32b VLSI Micromainframe Computer System" ISSCC '81 Digest, pp. 110-111.
19. Alan B. Oppenheim, "Applications of Digital Signal Processing". Prentice-Hall, Inc [1978].
20. Abraham Peled-Dede Liu, "Digital Signal Processing". John Wiley and Sons. [1976].
21. Lawrence R. Rabiner-Bernard Gold, "Theory and Application of Digital Signal Processing". Prentice-Hall [1975].
22. W.S. Richardson et al, "The 32b Computer Instruction Decoding Unit". ISSCC '81 Digest, pp. 114-115.
23. Lawrence Snyder, "Introduction to the Configurable, Highly Parallel Computer". IEEE Computer, Jan 1982, pp. 47-56.
24. Harold S. Stone, "Introduction to Computer Architecture". S.R.A., [1975].
25. Ian Watson and John Gurd, "A Practical Data Flow Computer". IEEE Computer, Feb 1982, pp. 51-57.
26. Jack B. Dennis, "Data flow supercomputers" IEEE Computer, Nov. 1980, pp. 48-56.
27. Masaaki Iwadachi et al., "A 6 NS, 4 KB bipolar RAM using switched load resistor memory cell" ISSCC-79 Digest, pp. 108-109.
28. Atsuo Hotta et al., "Bipolar memory LSI chips for computers" ISSCC-79 Digest, pp. 98-99.
29. Douglas Johnson, "Data flow machines threaten the program counter" Electronic Design, Nov. 22, 1980, pp. 225-258.
30. H.T. Kung, "Why Systolic architectures?" IEEE Computer, Jan. 1982, pp. 37-46.

1

2 2
OF / DE

31. D.F. Farmer et al, "A multicriteria approach to super-system architecture definition". IEEE Transactions on computers, May 1982, pp. 410-418.
32. J. McGraw, "Data flow computing: Software development". IEEE Transactions on computers, Dec. 1980, pp. 1095-1103.
33. G. Zick, "Expression processor: A pipelined multiple processor architecture". IEEE Transactions on computers, Aug. 1981, pp. 525-536.
34. A.P. Reeves, "A systematically designed binary array processor". IEEE Transactions on computers, April 1980, pp. 278-287.
35. T.Y. Feng, "On a class of multistage interconnection networks". IEEE Transactions on computers, Aug. 1980, pp. 694-702.
36. H.C. Torng, "On the analysis of memory conflicts and bus contentions in a multiple microprocessor system". IEEE Transactions on computers, Jan. 1979, pp. 28-37.
37. S.B. Wu, "A cluster structure as an interconnection network for large multimicrocomputer systems". IEEE Transactions on computers, April 1981, pp. 254-263.
38. D. Nassimi, "Data broadcasting in SIMD computers". IEEE Transactions on computers, Feb. 1981, pp. 101-106.
39. Richard P. Hopkins et al, "Data driven and demand driven computer architecture". ACM computing surveys, Vol. 14, No. 1, March 1982, pp. 93-143.

END

05109184

FIN