1

CANADIAN THESES ON MICROFICHE

THÈSES 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 an inferior 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.

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 qualité inférieure.

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.

THIS DISSERTATION
HAS BEEN MICROFILMED
EXACTLY AS RECEIVED

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

National Library
of Canada

Bibliothèque nationale
du Canada

CANADIAN THESES
ON MICROFICHE

0-315-19943-1

THÈSES CANADIENNES
SUR MICROFICHE

42

67850

NAME OF AUTHOR/NOM DE L'AUTEUR KOSTAS SIDMALAS
TITLE OF THESIS/TITRE DE LA THÈSE SYNTHESIS OF OPTIMAL ARCHITECTURES FOR
IMPLEMENTING DIGITAL SIGNAL PROCESSING OPER-
RATIONS
UNIVERSITY/UNIVERSITÉ CARLETON UNIVERSITY
DEGREE FOR WHICH THESIS WAS PRESENTED/
GRADE POUR LEQUEL CETTE THÈSE FUT PRÉSENTÉE Ph. D.
YEAR THIS DEGREE CONFERRED/ANNÉE D'OBTENTION DE CE GRADE 1985
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 other-
wise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈ-
QUE 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.

DATED/DATE 4/9/85 SIGNED/SIGNÉ K. Sidmalas

PERMANENT ADDRESS/RÉSIDENCE FIXE NAVARINO 18, G.R. 155-62 CHOLARGOS,
ATHENS, GREECE

**SYNTHESIS OF OPTIMAL ARCHITECTURES FOR IMPLEMENTING
DIGITAL SIGNAL PROCESSING OPERATIONS**

by

Konstantinos O. Siomalas

This Thesis was submitted to the Faculty of Graduate
Studies in partial fulfillment of the requirements for the
degree of Doctor of Philosophy.

Department of Systems and Computer Engineering

Carleton University

Ottawa, Ontario

November 1984

© copyright

1985, K. O. Siomalas

The undersigned recommend to the Faculty of Graduate
Studies acceptance of the thesis:

Synthesis of Optimal Architectures for Implementing Digital
Signal Processing Operations

Submitted by K.O.Siomalas, B. Eng., M. Eng., in partial
fulfillment of the requirements for the degree of Doctor of
Philosophy.

Thesis Supervisor

Chairman,
Department of Systems and
Computer Engineering

CARLETON UNIVERSITY

February 1985

TABLE OF CONTENTS

	<u>Page</u>
ABSTRACT.....	1
ACKNOWLEDGEMENTS.....	2
CHAPTER 1. Introduction	
1.1 Digital Signal Processing.....	3
1.2 The Subject of the Thesis.....	4
1.3 Organization of the Thesis.....	5
CHAPTER 2. DSP Algorithms and Computations	
2.0 Introduction.....	7
2.1 Structure and Representation of DSP Algorithms.....	7
2.2 Exploiting Paralellism in DSP Algorithms..	10
2.3 Summary.....	19
2.4 References.....	21
CHAPTER 3. Systems' Architecture	
3.0 Introduction.....	22
3.1 Architectural Considerations in Signal Processing Design.....	23
3.2 Simulation	
3.2.1 Introduction.....	24
3.2.2 Algorithm Encoding.....	25
3.2.3 Simulation.....	31
3.2.4 Usage.....	35
3.2.5 Summary.....	37
3.3 Synthesis of Efficient Architectures	
3.3.1 Introduction.....	38
3.3.2 Analysis.....	40
3.3.3 Conclusions.....	74
3.4 Summary.....	75
3.5 References.....	77

	<u>Page</u>
CHAPTER 4. Performance Constraints in VLSI Implementation	
4.0 Introduction.....	80
4.1 Processing Element Multiplicity and Pin Limitations	
4.1.1 Introduction.....	80
4.1.2 Performance of VLSI Systems with Single-Input and Single-Output Ports.	82
4.1.3 VLSI Systems with a Time Multiplexed I/O Port	
4.1.3.1 Introduction.....	90
4.1.3.2 Simple Models.....	91
4.1.3.3 General Models.....	97
4.1.3.4 Applications.....	105
4.2 Summary and Conclusions.....	114
4.3 References.....	116
CHAPTER 5. Conclusions.....	118
APPENDIX	
Chapter 3.....	123
Chapter 4.....	132

LIST OF FIGURES

CHAPTER 2

Figure 2.1 : Hardware Representation of an FFT Butterfly Algorithm.

Figure 2.2 : a) Data Flow Graph; b), c) Corresponding Control Flow Graphs, and d) Precedence Table.

CHAPTER 3

Figure 3.1 : 3-section Nonrecursive Filter Algorithm.

Figure 3.2 : a) Two-pole Filter Algorithm;
b) A Hypothetical Algorithm.

Figure 3.3 : A Sample Run for a 4-point FFT

Figure 3.4 : The Cost/Performance Ratio as a Function of Multipliers and Adders.

Figure 3.5 : (a) A General Structure of a DSP System
(b) Structure of the Processing Unit.

Figure 3.6 : (a) Timing for One Execution of a Type 1, n-level Algorithm.
(b) Timing for Repeated Executions of the same Algorithm.

Figure 3.7 : (a) Timing for One Execution of a Type 2, n-level Algorithm.
(b) Timing for Repeated Executions of the Same Algorithm.

Figure 3.8 : Definition of x_i , z_i and d_i .

Figure 3.9 : (a) Decimation in Time Butterfly Flow Graph
(b) The Corresponding Hardware Algorithm.

Figure 3.10: (a) Decimation in Frequency Butterfly Flow Graph.
(b) The Corresponding Hardware Algorithm.

Figure 3.11: Execution Timing for the Butterfly Processors
(a) DIT.
(b) DIF.

Figure 3.12: 8-point FFT Algorithm.

Figure 3.13: Execution Timing for the N-point FFT Processor.

CHAPTER 4

Figure 4.1 : Block Diagram of a Single-Input Single-Output Port VLSI System, Consisting of Identical PEs.

Figure 4.2 : Timing for the Case of $t_{tr} < t_p$.

Figure 4.3 : Timing Diagram for the Case of $t_{tr} > t_p$.

Figure 4.4 : Timing Diagram Used for the Proof of Theorem 1

Figure 4.5 : Timing Diagram used for the Proof of Theorem 2

Figure 4.6 : Timing Diagram for the MAC-Chips [7].

Figure 4.7 : Structure of a System Consisting of Several PEC and DTC Operators.

Figure 4.8 : Timing Diagrams; (a) $t_{tr} \geq t_p$.
(b) $t_{tr} < t_p$.

Figure 4.9 : Timing Diagram for the Case of $t_i = 0$ (Case 1).

To my parents

Ὁρέστης and Ἀσπασία

**SYNTHESIS OF OPTIMAL ARCHITECTURES FOR IMPLEMENTING
DIGITAL SIGNAL PROCESSING OPERATIONS**

"... πρὸς στάθμην πέτρον τίθεσθαι, μὴ
τι πρὸς πέτρῳ στάθμην.."

Πλούταρχος

"... adjust the stone to fit the line,
and not the line to fit the stone.."

Plutarch

ABSTRACT

This work deals with the design of very high performance and highly efficient digital signal processing (DSP) systems.

In the first part, a theoretical study is presented of the synthesis of the most efficient pipelined processing system for executing a DSP algorithm. This is achieved by first determining the maximum parallel form of the algorithm and then allocating the various arithmetic and data transfer operations of the algorithm to the processing elements of the system in such a manner that all basic components are concurrently active. Parallel processing at each stage of the pipeline is used to achieve the best possible performance. The model is also used to demonstrate how the overall problem can be reformulated as an optimization problem, which minimizes a cost function of the system under certain constraints.

The final part of the thesis demonstrates the relationship between data transfers through shared ports and the performance of the processing elements of the system. From this relationship a balance can always be achieved so as to increase the multiplicity of the processing elements (and hence the performance) to the point where no I/O conflicts occur.

ACKNOWLEDGEMENTS

I would like to offer my appreciation to the members of the defense committee (Dr. Stuart Tewksbury - Bell Laboratories NJ, Prof. C. Chan - Department of Electronics, Prof. D.D. Falconer, and Prof. R.J.A. Buhr) for their effort in reading the thesis manuscript. I would like also to express special thanks to my thesis supervisor Prof. B.A. Bowen for his innumerable contributions to this dissertation.

This work was supported in part by the Defense Research Establishment, Ottawa.

K. O. Siomalas

November 1984

CHAPTER 1. Introduction

1.1 Digital Signal Processing

Digital Signal Processing (DSP) is concerned with the representation of signals by sequences of numbers and the processing of these sequences. Such processing may be used to estimate characteristic parameters of a signal or to transform a signal into a more desirable form.

The field of DSP has grown enormously in the past two decades due to the availability of high performance digital computers which have boosted the development of increasingly complex and sophisticated DSP algorithms. Moreover, recent advances in VLSI technology promise economical implementations of very complex DSP systems.

DSP for the most part relies on the theory of discrete-time linear time-invariant systems, and it is mainly divided into digital filtering and spectrum analysis. It is certain, however, that almost all the theoretical considerations involved in the design of digital filters and spectrum analyzers would be of little value if a good understanding of those issues involved in practical implementations of such systems in software or hardware, were not also available.

1.2 The Subject of the Thesis

Until recently, DSP has mostly been carried out using general purpose digital computers. The processing, however, could not always be done in real time and most of the applications were limited to approximate or simulate an analog signal processing system. With the advent of very large scale integration (VLSI), special purpose components were developed that made possible the implementation of digital signal processors (digital filters, FFT processors, etc) with sampling rates in the megahertz range. Due to real time requirements and the great amount of computation required to carry out a DSP algorithm, optimized processor architectures are needed to overcome speed limitations imposed by individual components.

The field of digital signal processing appears to be lacking in a general processor optimization technique. Efforts have been made to optimize DSP algorithms (by minimizing multiplications for example, as in the FFT), since algorithms directly affect both the performance and the cost of a processor. However, even an optimized algorithm may be intrinsically hostile, in terms of the execution requirements to the computational capabilities of the processor. In this event, even systems with great processing capability would still yield poor performance, and become inefficient.

The present work presents a theoretical study and results which form the basis of a methodology for the design of efficient and very high performance digital signal processing systems.

There are two main problems that will be considered in this work. The first problem deals with the optimal partition and allocation of operations in a DSP algorithm to the processing elements of a processor, to obtain the most efficient processing system. A combination of parallel-pipelined processing will be used to ensure the highest possible performance.

A major implementation constraint of parallel or pipelined systems in the VLSI environment, is the number of pins available on a chip. In the second phase of the thesis, we shall attempt to formalize and provide answers to the problem of determining the maximum multiplicity of parallel operating processing elements that can be realized on a single VLSI chip, and share a single I/O port without conflict.

1.3 Organization of the Thesis

The work is mainly divided in three parts. The first part, (Chapter 2), deals with the structure of DSP algorithms. The first step in the design of a multiple element digital signal processor specialized for a

specific DSP algorithm, is to ensure that the computational algorithm is in the most efficient form for parallel or pipelined processing. These processing techniques yield high execution rates when used by a multiple-processing system. A method for determining the maximum parallel form of a DSP algorithm is presented in Section 2.2 .

The second part, (Chapter 3), consists of the main study which provides the means for determining the structures of the most efficient (optimal) DSP systems.

The highly modular structure that characterizes the proposed architectures of Chapter 3, makes it technologically feasible to build the whole system on a high density printed circuit board or even on a single VLSI chip. In the third part of the thesis, (Chapter 4), some issues and results are presented, related to VLSI implementation of DSP systems.

Note that the literature review on publications related to this work, is done in the introduction of the sections 3.3 and 4.1. Also, the references are provided at the end of each chapter.

CHAPTER 2. DSP Algorithms and Computations

2.0 Introduction

The careful consideration of the algorithms to be executed is essential in the design of any high performance specialized DSP processor. The examination of a DSP algorithm can show what structure, flexibility, and functions are required in the architecture of the processor. The characteristics, as well as, the representation of DSP algorithms is the subject of Section 2.1.

Most DSP algorithms exhibit a substantial amount of parallelism which must effectively be exploited in order to achieve the highest possible performance. Section 2.2 deals with this problem, namely, evaluating the inherent parallelism in DSP computational structures.

2.1 Structure and Representation of DSP Algorithms

Digital signal processing algorithms are based in the mathematics of difference equations and they can always be written as a collection of algebraic expressions involving operands and operators. Usually,

the operators are composed of simple arithmetic operations, such as, multiplications and additions.

DSP algorithms are basically procedures for the transformation of one sample sequence (input data) into another (output data). Since the data sequences are almost always regular and deterministic, the processing can be divided into a fixed number of steps that can be executed in a predetermined sequence by a set of processing elements (PEs).

The representation of a DSP algorithm may take many forms. Algebraic representations, however, may fail to offer insight into the behavior of an algorithm. In the design of a DSP processor algorithm representations are needed that will offer information about memory accesses, I/O, as well as, the structure of arithmetic operations. The flow graph notation [1,3,4] provides enough information about the algorithm it represents, and, it can easily be transformed into the more flexible hardware representation used in this work.

The hardware representation of an algorithm (see Figure 2.1) is a form of a signal flow graph where each node represents an arithmetic operation, and each edge represents an item of data or a data transfer operation. Data transfer operations may also include storage (delay) functions, and/or, I/O.

Figure 2.1 : Hardware representation of an FFT Butterfly Algorithm

The exact sequence of operations needed to realize an algorithm is not unique and some means for determining the most efficient form of the algorithm is required. This will be the main subject of the next section.

2.2 Exploiting Parallelism in DSP Algorithms

Most of signal processing requirements call for repetitive execution of certain algorithms within an available dwell period. Therefore, real-time DSP systems must be able to organize and process a great amount of incoming data in minimal time. Control mechanisms, as well as computing speed, are the major decisive factors on system's performance. Establishing precedence relationships of operations within an algorithm is important, because it is a key step in preparing the control program segment that will appear repeatedly in the structure of entire application. Most DSP algorithms exhibit a substantial amount of parallelism which must be effectively exploited in order to achieve the highest performance.

A general model based on computation structures, and algorithms to extract parallelism is described in [2]. A simplified approach applicable to data independent algorithms only (such as most DSP algorithms), based on the model in [2] is presented in this section. Note

that all the definitions and the expressions needed for the modeling of a computation, have been taken from reference [2]. The two theorems and the algorithms to be presented are new. The results obtained fit our conceptual representation of DSP algorithms, and are simpler to apply and use.

The algorithms to be processed are completely described by a computation structure. A computation structure consists of two parts, control and data, each represented by a directed graph called a precedence graph and a data flow graph, respectively [2]. A data flow graph consists of a set of nodes (containing storage cells and operators) and a set of edges indicating the input and output storage cells needed by the operators. An operation can be defined as $Fa([A], [B]) \rightarrow D$. This means that the content of cell A and content of cell B are operated upon by operator a and the result is stored in cell D.

It is assumed that all operators are functional (have an output cell(s)). An AND operator A is used to synchronize parallel flow of control in which the flow can proceed to the next operator(s) only when all operations prior to the AND operator have been completed. A precedence graph provides information about the execution ordering of the various operations (operators), thus indicating the desired flow of control. A data

expression DE_i is defined to be a set that specifies the input and output data cells required for operator i .

$DE_a = [I_a, O_a]$ where

I_a is the set of input data cells

O_a is the set of output data cells

Example: For the data flow and control flow graphs in Figure 2.2 is

$DE_a = [(A, B), D]$

$DE_b = [(B, C), E]$

$DE_c = [(D, E), F]$

From a computational point of view, the two control flow graphs (b) and (c) are equivalent. However, (b) uses maximum parallelism; therefore it can be executed faster than the strictly serial (c).

The following relations are defined for a pair of operators (a, b) in a computation

$a // b$ a, b are independent, i.e.,

$$O_a \cap (I_b \cup O_b) = \emptyset$$

$$O_b \cap (I_a \cup O_a) = \emptyset$$

	a ₁	a ₂	a ₃
a ₁	$\times$	$\wedge$	$>$
a ₂	$\wedge$	$\times$	$>$
a ₃	$<$	$<$	$\times$

(d)

Figure 2.2 : a) Data Flow Graphs; b), c) Corresponding Control Flow Graphs, and d) Precedence Table

In other words, the destination cell of the result of operation a, is neither the source, nor the destination cell of operation b, and conversely

$a > b$ a must precede b for correct operation

$a < b$ b must precede a for correct operation

$a \wedge b$ a, b in parallel, i.e., it is neither $a > b$, nor $a < b$

The following two theorems are used in the algorithms to be described later.

Theorem 1: The operations $A_1, A_2, \dots, A_n$ can be performed in parallel iff:

$A_1 \wedge A_2, A_1 \wedge A_3, \dots, A_1 \wedge A_n$

$A_2 \wedge A_3, A_2 \wedge A_4, \dots, A_2 \wedge A_n$

$\dots$

$A_{n-1} \wedge A_n$

Proof: Obvious

Theorem 2: Let $A = [A_1, A_2, \dots, A_n]$ and $B = [B_1, B_2, \dots, B_m]$ be parallel sets (i.e., $A_1 \wedge A_2 \wedge \dots \wedge A_n$ and $B_1 \wedge B_2 \wedge \dots \wedge B_m$) then $A > B$, i.e., A must precede B for correct operation, iff:

$A_i > B_j$ or $A_i \wedge B_j$ for all i, j

Proof: Obvious

For any computation, we can represent the operator relations $>$ or $<$ (precedence), $\wedge$ (parallel) with a precedence table P , a square $n \times n$ matrix with a unique entry for each operator pair (a_i, a_j) , where a_i, a_j are the row and column designation, and $i \neq j$.

Algorithm for determining max parallelism:

STEP 1: Assign an operator name for each operation of the given algorithm and draw the required data-flow graph.

STEP 2: Prepare a precedence table P , as described above and specify its entries as in the following page :

```

REPEAT FOR ALL i,j, i≠j
    IF P(ai,aj) = ∅ (not yet specified)
        AND  $O_{ai} \cap (I_{aj} \cup O_{aj}) = \emptyset$ 
        AND  $O_{aj} \cap (I_{ai} \cup O_{ai}) = \emptyset$ 
    THEN specify P(ai, aj) = //
    ELSE IF  $O_{ai} \cap I_{aj} \neq \emptyset$ 
        THEN P(ai, aj) = >
        ELSE IF  $O_{aj} \cap I_{ai} \neq \emptyset$ , THEN P(ai, aj) = <
    END IF
    IF P(ai, aj) = < (or >), AND
        P(aj, ak) = < (or >) THEN
            P(ai, ak) = < (or >)
    END IF
REPEAT FOR ALL i,j, i≠j
    IF P(ai, aj) = // THEN P(ai, aj) = ^
END IF

```

(Method-i)

STEP 3a: For each row of P determine the maximum number of parallel relations by applying Theorem 1.

STEP 3b: Choose the row (or rows) with the largest number of parallel relations and determine the corresponding parallel set(s).

(Method-ii)

STEP 3: Choose the row (or rows) with entries the relations $>$ and/or $\wedge$ only. The corresponding operators consist of a parallel set (verify that, by applying Theorem 1). If there is not such a row, the operators of the remaining rows consist of a parallel set, as well.

STEP 4: Delete all rows and columns of P , corresponding to the operators contained in the parallel set found in (3) and go to (3). (Repeat until all operators have been grouped in parallel sets).

STEP 5: Determine the precedence of parallel sets by applying Theorem 2.

The above algorithms can be used to decompose a large DSP algorithm (consisting of several basis function operations) into as many as possible small parallel executable segments; thus the time required to execute the algorithm can be reduced to a minimum.

Example: Parallelism analysis of the FFT butterfly (for simplicity only arithmetic operations are used).
Let (see Figure 2.1)

$$\begin{aligned} M1 &= X2 \cos a \\ M2 &= Y2 \sin a \end{aligned}$$

$$A1 = M1 + M2$$

$$M3 = X2 \sin a$$

$$M4 = Y2 \cos a$$

$$A2 = M4 - M3$$

$$A3 = X1 + (M1 + M2)$$

$$A4 = X1 - (M1 + M2)$$

$$A5 = Y1 + (M4 - M3)$$

$$A6 = Y1 - (M4 - M3)$$

The precedence table P is as follows:

	M1	M2	A1	M3	M4	A2	A3	A4	A5	A6
M1	X	$\wedge$	$>$	$\wedge$	$\wedge$	$\wedge$	$>$	$>$	$\wedge$	$\wedge$
M2	$\wedge$	X	$>$	$\wedge$	$\wedge$	$\wedge$	$>$	$>$	$\wedge$	$\wedge$
A1	$<$	$<$	X	$\wedge$	$\wedge$	$\wedge$	$>$	$>$	$\wedge$	$\wedge$
M3	$\wedge$	$\wedge$	$\wedge$	X	$\wedge$	$>$	$\wedge$	$\wedge$	$>$	$>$
M4	$\wedge$	$\wedge$	$\wedge$	$\wedge$	X	$>$	$\wedge$	$\wedge$	$>$	$>$
A2	$\wedge$	$\wedge$	$\wedge$	$<$	$<$	X	$\wedge$	$\wedge$	$>$	$>$
A3	$<$	$<$	$<$	$\wedge$	$\wedge$	$\wedge$	X	$\wedge$	$\wedge$	$\wedge$
A4	$<$	$<$	$<$	$\wedge$	$\wedge$	$\wedge$	$\wedge$	X	$\wedge$	$\wedge$
A5	$\wedge$	$\wedge$	$\wedge$	$<$	$<$	$<$	$\wedge$	$\wedge$	X	$\wedge$
A6	$\wedge$	$\wedge$	$\wedge$	$<$	$<$	$<$	$\wedge$	$\wedge$	$\wedge$	X

Note that the relation $A1 > M1$ is equivalent to $M1 < A1$.

The first parallel sets found after (3) are (M1, M2, M3, M4) and (A3, A4, A5, A6).

After step (4) the table becomes:

	A1	A2
A1	X	$\wedge$
A2	$\wedge$	X

Therefore the remaining parallel set is (A1,A2). After executing (5), the precedence of the three sets found to be (M1, M2, M3, M4) (A1, A2) (A3, A4, A5, A6). The control expression is $CE = (M1 \wedge M2 \wedge M3 \wedge M4) (A1 \wedge A2) (A3 \wedge A4 \wedge A5 \wedge A6)$. At this point, it must be noted that there is no intent to suggest that this is a practical way to analyze an algorithm as simple as the FFT butterfly, in which the parallel operations are rather obvious; it is only intended to show how the method works. This method becomes very useful when a long and complicated DSP algorithm has to be analyzed for parallelism.

2.3 Summary

The most important representation of DSP algorithms in the context of the present work, is a form suitable for translating them into programs that will reside in the signal processor memory. The hardware representation presented in this chapter not only highlights the architectural requirements for a signal processor,

but also enables someone to actually determine the execution sequence of operations for the resulting processor.

The main issue of this chapter was the method for organizing the operations in a DSP algorithm into the most efficient form for parallel computation. Models for matching algorithms to architectures based on efficiency optimization, and a systematic processor design approach, are presented in the following chapter.

2.4 References

- [1] L.R.Rabiner and B.Gold, "Theory and Applications of Digital Signal Processing", Prentice-Hall 1975.
- [2] M.C.Wei and H.A.Sholl, "An Expression Model for Evaluation of Parallelism in Control Structures", IEEE Trans. Computers, Vol. C-31, No.9, pp851-863, Sept. 1982.
- [3] J.S.Thomson and S.K.Tewksbury, "LSI Signal Processor Architecture for Telecommunication Applications", IEEE Trans. Acoust., Speech, Signal Processing, Vol. ASSP-30, No.4, pp613-631, Aug. 1982.
- [4] S.Y.Kung, "On Supercomputing with Systolic Wavefront Array Processors", Proc. IEEE, Vol.72. No.7, pp867-884, July 1984.

CHAPTER 3. Systems' Architecture

3.0 Introduction

The computational requirements of DSP algorithms are often too severe to be met by any but the largest and most expensive general purpose computing systems, especially in real-time applications. As a result, many applications of DSP techniques can only be realized by the design of special purpose processors. In general, a special purpose processing system could be designed that would meet any given performance requirement by employing large quantities of dedicated hardware. Since signal processing is arithmetic intensive the efficient use of such complex arithmetic structures is of primary importance.

This chapter deals with the design of the processing units, (arithmetic units), of specialized DSP processors, namely, systems designed to implement a specific DSP algorithm.

The general architectural characteristics of a DSP processor are discussed in Section 3.1. Along with the analytical results of the study of the synthesis of efficient processing structures presented in Section 3.3, some simulation results are also presented in Section

3.2, to demonstrate the trade-offs associated with processing element multiplicity, efficiency, and the processing power of the computing system.

3.1 Architectural Considerations in Signal Processing Design

In most real-time signal processing applications there is a need for performance above that deliverable by simple hardware components. Pipelining and parallel processing techniques can increase the computation rate of a digital system [15]. The structure of most DSP algorithms, and the simple and regular data flows of DSP operations make possible the use of pipelining, as well as, parallel processing at each state of the pipeline, so that the best possible performance can be achieved.

There are many important issues in designing these massively parallel processor systems, such as, processor interconnection, synchronization and modularity. Interconnection is the most critical issue of the design, since communication is very expensive in terms of area, power, and time; therefore, localized interconnections must be utilized. For this reason, local and regular data movements are strongly preferred.

Synchronization is also a critical issue in the design. For large systems both global and self-timed

schemes must be utilized to ensure the highest possible clock rate [16].

VLSI implementation requirements suggest using repetitive modular structures, namely, a few different types of simple processing elements (modules).

In the remaining sections of this chapter we shall see how these highly concurrent structures can be exploited to achieve faster speeds and high efficiency in computing operations, ranging from the most elementary to quite complicated ones.

3.2 Simulation

3.2.1 Introduction

This section presents results obtained by a program which simulates the control operations of the arithmetic unit of various basis function signal processors. The simulation deals with the control sequence required for arithmetic operations only. For simplicity, data flow control will not be considered in the simulation.

DSP basis function algorithms are composed mostly of multiplications and additions. The simulation program was written to simulate processors executing this type of algorithm. The program accepts information about the structure of the DSP algorithm to be processed, the architecture of the particular processor, namely the

number of multipliers and adders, and prints out the operation allocation mapping, the execution time in cycles, the minimum number of required adders, as well as the cost/performance ratio of the particular processor.

The purpose of this simulation was to check the validity of the analytical results that will be presented in the next section, as well as, to explore the performance and the associated trade-offs, of various processors executing certain DSP algorithms.

3.2.2 Algorithm Encoding

This subsection describes an algorithm encoding technique developed specifically for the processor simulation program which is described in the next section.

In order to simulate the operation of a signal processor executing an algorithm, the structure of the algorithm must be known by the simulator. The technique described here encodes the structure of the algorithm in an easy to program code. The following assumptions have been made concerning the structure of the algorithms:

- i) The algorithm consists of several levels of one or more parallel operations;
- ii) The first (top) level consists strictly of multiplications, and the subsequent levels consist strictly of additions (there is no distinction between additions and subtractions). Note that this assumption is valid for some algorithms only.

An algorithm matrix has been introduced describing the precedence relations of the algorithm. Each element of this matrix indicates a number of parallel operations, performed on a particular level of the algorithm. There are as many columns as the levels of the algorithm, and as many rows as multiplications. The elements of the first column, arranged in descending order, indicate the number of performed multiplications, at a given time. All the other columns indicate the maximum number of operations that can be performed on each level, when the execution of operations indicated by the elements of the previous columns has been completed.

Examples

- 1). FFT Butterfly

The well known structure of this algorithm is shown in Figure 2.1 . It is a 3-level algorithm with 4 multiplications; therefore the algorithm matrix is a 4x3 matrix with elements as follows:

4	2	4
3	1	2
2	1	2
1	0	0

According to this algorithm 4 multiplications are needed to perform 2 additions of the second level. After the 2 additions have been completed, another 4 additions (third level) must be performed. This precedence relation is indicated by the first row of the matrix (4-2-4). The second row indicates the precedence when 3 multiplications have been completed. In this case only one addition of the second level can be performed, which in turn triggers the execution of the two additions of the third level, and so forth. Note that without the last column, this matrix is identical to complex multiplication algorithm matrix.

2) Three-section Nonrecursive Filter

The structure of this algorithm is shown in Figure 3.1. The algorithm matrix is as follows:

4	1	1	1
3	1	1	0
2	1	0	0
1	0	0	0

3) Two-pole Filter

Figure 3.2(a) shows the algorithm structure of a two-pole filter. In this case the algorithm matrix is:

2	1	1
1	0	0

4) A Hypothetical Algorithm

For a better understanding of the method, the structure of a hypothetical algorithm, shown in Figure 3.2(b), has been chosen. The algorithm matrix for this case is as follows:

Figure 3.1 : 3-section Nonrecursive Filter Algorithm

(a)

(b)

Figure 3.2 a) Two-pole Filter Algorithm;
b) A Hypothetical Algorithm

6	2	1	1	2
5	2	1	0	0
4	2	1	0	0
3	1	0	0	0
2	1	0	0	0
1	0	0	0	0

At this point, it is important to note that an erroneous algorithm matrix will result in a false output of the simulation program, or even an aborted execution. Therefore the user must be extremely cautious in the determination of the algorithm matrix.

3.2.3 Simulation

The program to be presented in this subsection simulates the control operations of the arithmetic unit of various basis function signal processors. The simulation deals with the control sequence required for arithmetic operations only. Data flow control will not be considered at the present.

Description

1) Inputs

a) Inputs concerning the algorithms

The program accepts information about the structure of the algorithm in the form of the algorithm matrix described in the previous section. The algorithm matrix along with its dimensions (number of levels and number of lines), and the required number of executions, is the only information needed about the algorithm.

b) Inputs concerning the processor

The number of available multipliers must be entered, and in case it is required, the number of adders as well; otherwise the program itself will compute the maximum number of required adders.

2) Outputs

The program will print out the mapping for the allocation of the arithmetic operations of each cycle, the number of cycles required for the execution, as well as the cost/performance ratio of the particular processor.

3) Arrays

The following 2-dimensional arrays are being used in the simulation program.

MUL: This array, at the end of the simulation, will contain the allocation mapping for the multiplications. Each row indicates the multiplications required for each execution of the algorithm, and each column indicates overlapped multiplications in each cycle.

AD: This is a similar array to MUL but it indicates additions.

ALG: This array contains the algorithm matrix described in the previous section.

AUX: This is an auxiliary array for internal use. It is used to store the intermediate results in each level of the algorithm. At the end of each execution all the rows of this array are summed up to produce a row of the AD matrix. The AUX matrix has as many rows as the levels of the algorithm to be processed.

More details for the structure of programs can be found in the documentation of the program in Appendix I. The dimensions of the various arrays can be easily modified. The dimensions of the program in Appendix I require a 57K main memory, and the program can be used for up to 65 executions of the same algorithm. A sample

A NEW ALGORITHM ? YES=0, NO=1,....9.
?0

HOW MANY LEVELS ?
?3

ENTER NO OF LINES
?4

ENTER ALGORITHM'S MATRIX
?040204
?030102
?020102
?010000

ENTER NO OF EXECUTIONS
?4

ENTER NO OF MULTIPLIERS
?3

LIMIT ON AVAILABLE ADDRESS YES=0, NO=1,....9.
?1

ALLOCATION PRINTOUT YES=0, NO=1,....9.
?0

```

0 1 3 2 0 0 0 0
0 0 1 3 2 0 0 0
0 0 0 0 2 4 0 0
0 0 0 0 0 1 3 2

```

ADDINS

MLTFLS

```

3 1 0 0 0 0
0 2 2 0 0 0
0 0 1 3 0 0
0 0 0 0 3 1

```

4 EXECUTIONS IN 8 CYCLES, .50 EXECUTIONS/CYCLE
AT LEAST 5 ADDERS REQUIRED
COST/PERFORMANCE RATIO : 16.00

CONTINUE ? NO=0, YES=1,....9
?0

STOP

Figure 3.3: A Sample Run for a 4-Point FFT

run of the program, for a 4-point FFT, is given in Figure 3.3.

3.2.4 Usage

This subsection presents some issues on the design of DSP processor ALUs using as design aid the described simulation program.

The main aim in the design of a DSP processor is to achieve performance with as low a cost as possible. The proposed simulation program can be used to determine the performance and the efficiency of a signal processor's arithmetic unit, which is to execute one or more basis function algorithms. Also it is possible to determine the minimum control program segment that will appear repeatedly during the execution of a certain basis function algorithm.

Example: FFT-Processor

The efficiency (C/P ratio) of various ALUs, consisting of multiple multipliers and adders, and executing a 16-point FFT algorithm, is shown in Figure 3.4, which illustrates plots of the cost/performance ratio as a function of the number of adders and

Figure 3.4 : The Cost/Performance Ratio as a Function of Multipliers and Adders

multipliers, for these FFT processors. The C/P ratio has been calculated by using the following formula:

$$\frac{C_m N_m + C_a N_a}{(\# \text{ of executions}) / (\# \text{ cycles})}$$

where C_m = cost of a multiplier, C_a = cost of an adder,
 N_m = number of multipliers, N_a = number of adders.

Hence we have assumed that $C_m = C_a = 1$. It is obvious from the graph that the minimum value of the C/P ratio occurs for a particular ratio of the basic processing elements, (in this case multipliers and adders), of the system.

3.2.5 Summary

This section presented a useful design tool that can be used in the early stages of the design of a DSP processor. The presented program simulates the sequence of the arithmetic operations required to perform any DSP algorithm, and it gives a clear view of the operations within the arithmetic unit of the processor. It is evident from the presented results that there

exists an optimum combination and allocation of the operations in an algorithm to the processing elements of the processor, such that, the efficiency of the system is maximized. An attempt to establish this optimality theoretically will be presented in the next section.

3.3 Synthesis of Efficient Architectures

3.3.1 Introduction

This section presents a theoretical study and results which form the basis of a methodology for the design of efficient specialized processing units, which may consist of a general purpose digital signal processor.

Several papers have appeared in the literature which deal with optimal scheduling strategies in multiprocessor systems [1], [2], [3]. Most of the research is oriented towards general distributed processing systems in a data processing environment. However, systems used for signal processing exhibit several unique features by comparison to other processing systems. In digital signal processing, data sequences are almost always regular and deterministic. The processing steps can be divided into a fixed number and can be executed in a predetermined sequence in a set of processing elements. Real-time requirements existing in most applications, demand concurrent data acquisition, processing and data output activities. There have been several papers in the

literature that deal with processor architectures for real-time signal processing. Most of them deal with the design of specific DSP processors, such as FFT processors [4, 12, 13, 14], Convolution [7], and other special purpose devices. More general approaches to the design of DSP processors have been presented in some publications [6, 8, 9, 10]. The architectural characteristics of DSP systems are discussed in detail in [9] and [10]. In [6] the problem of designing a distributed DSP processor is examined from the point of view of efficient use of system processing resources. Several assumptions in this paper have limited the applicability of the results to DSP algorithms consisting of high level operations only (i.e., algorithms consisting of several FFTs, filtering, etc.). In [8] several new parallel architectures are presented based on Cordic processors, for the solution of some common problems in signal processing and matrix algebra.

The present work presents a different approach based on highly efficient architectures characterized by a combination of pipelining-parallel processing, and an effort has been made to unify the theory and implementation to some degree.

3.3.2 Analysis

A. Processor Organization

The block diagram of a DSP system is depicted in Figure 3.5 (a). The processing unit does all the necessary operations of the algorithm, namely, arithmetic and data transfer operations (in signal processing the movement of data and coefficients is vitally linked to the arithmetic unit of the system). The control processor determines the precedence of the various operations in the system. The memory units store intermediate results, coefficients and the program segments of the control processor. Since signal processing is arithmetic intensive, the structure and the efficiency of the processing units is of primary importance, particularly in the case where many processing elements are utilized to meet high performance requirements. This work deals mostly with the design of very high performance processing units specialized for specific DSP algorithms. The structure of a processing unit is depicted in Figure 3.5(b). This structure has been chosen since it is ideal for pipeline-parallel processing and it appears to have a linear cost-performance relation as the number of stages increases [11].

(a)

(b)

Figure 3.5: (a) A General Structure of a DSP System
(b) Structure of the Processing Unit

B. Assumptions

The following assumptions are made concerning DSP algorithms, their processing by the processors, and the structure of the processing units.

- (1) Without loss of generality, we assume that the algorithms consist of n levels of operation, each level consisting of several basic operations that can be executed in parallel. In the case of a low level algorithm (basis algorithm), such as the FFT butterfly, basic operations are considered to be arithmetic operations (such as multiplications and additions), as well as single byte data transfer operations. In the case of a high level algorithm such as an N -point FFT algorithm, basic operations are considered to be a higher level arithmetic operation (such as FFT-butterfly) and multiple byte transfer operations.
- (2) A pipeline realization of such an algorithm consists of hardware stages, each stage consisting of several processing elements operating in parallel. In order to achieve as high performance as possible and keep the control complexity to a minimum, the number of pipeline states must be equal (with a few

exceptions only, depending on the algorithm) to the number of the algorithm levels [5].

- (3) We identify two different types of algorithms.
- Type 1: The algorithm is such that no operation of the i th level can be performed before the completion of all operations of the $(i-1)$ th level (see Figure 3.6). Type 2: The algorithm is such that some operations of the i th level can be performed before the completion of all operations of the $(i-1)$ th level (see Figure 3.7). Since the set of Type 1 algorithms is a subset of Type 2 algorithms in the subsequent analysis, we shall deal mostly with the Type 2 case.
- (4) The processing elements that consist of the segments (stages) of the pipeline are divided into two categories: i) Arithmetic operators, namely, processing elements that can perform one or more arithmetic operations (low or high level as defined in assumption 1) one at a time, and, ii) Data transfer operators (DT), namely, sets of hardware components, each set considered as a single processing element,

(a)

9

(b)

Figure 3.6 : (a) Timing for One Execution of a Type 1, n-level Algorithm
 (b) Timing for Repeated Executions of the Same Algorithm

Figure 3.7 : (a) Timing for One Execution of a Type 2, n-level Algorithm
 (b) Timing for Repeated Executions of the Same Algorithm

which can move data into or out of the processing unit.

- (5) The processing elements consisting of a segment of the pipeline can operate independently from each other. t_i is defined to be the time required by the operators of the i th segment of the pipeline to perform all the operations of the corresponding level of the algorithm. It is assumed that no operator will be continuously idle during the time t_i .
- (6) In the subsequent analysis we shall assume that all operators have the same execution rate of r ops (basic operations per second), either arithmetic or data transfer operations, depending on the operator. Although this is not the general case, in practice it is always possible to achieve a required speed by properly selecting the various hardware components of the system, and synchronize them with a common clock. In Section E we shall discuss this problem further.
- (7) We assume repeated execution of the algorithm by the processor (a typical case of DSP algorithm), on an input data set of M

different values arriving sequentially. It is assumed that in each particular segment (stage) of the pipeline, no operation on the next input data set can be performed before all operations on the last input set have been completed. This assumption simplifies both the theoretical analysis and the actual operation of the processor (otherwise a very complex control scheme would be required, for each pipeline segment, to keep track of an ever changing execution sequence) and we shall see it is good enough for an optimal allocation scheme.

C. Formulation

The definition of all the parameters and symbols used in the following analysis are given below:

n = The number of levels in the algorithm and/or the number of stages in the pipeline.

N_i = The number of parallel basic operations of the i th level of the algorithm.

σ = The total number of basic operations in the algorithm.

p_i = The number of operators of the i th stage (segment) of the pipeline.

r = The execution rate of an operator (see 6th assump.) in ops.

t_i = See assumption 5.

M = The number of executions of the algorithm.
Usually $M \rightarrow \infty$

t_e = Time required for M executions of the algorithm (execution time).

The timing for one execution of n -level algorithms is shown in Figures 3.6(a) and 3.7(a). Figures 3.6(b) and 3.7(b) show repeated executions of the same algorithms. The shaded areas show that no operation, on a particular level, is taking place at that time. This is due to bad synchronization, and/or utilization of greater computing power than that required, in some stages of the pipeline. The bigger those idle times in the execution, the more inefficient is the processing.

The time t_i , defined in assumption 5, depends only on the number of operations of level i and the number of operators of the corresponding stage of the pipeline. In other words

$$t_i = \left\lceil \frac{N_i}{rp_i} \right\rceil$$

We define the efficiency E of the processing unit as follows:

$$E = R / (\text{Total Computing Power in ops})$$

where R is the actual execution rate in ops, namely,

$$R = (\text{Total number of basic operations performed in } t_e) / t_e$$

The definition of the problem is as follows:

Given an n -level algorithm and a set of processing elements, find the optimum pipelined-parallel structure, by allocating the processing elements to levels of the algorithm in such a manner as to maximize the efficiency of the system.

The performance of the systems being considered here can be computed as follows:

Type 1 Algorithms

The execution time t_e of a Type 1, n -level algorithm can be derived by inspection of Figure 3.6(b), and it is:

$$t_e = t_1 + t_2 + \dots + Mt_m + t_{m+1} + \dots + t_n$$

or

$$t_e = \sum_{\substack{j=1 \\ j \neq m}}^n t_j + Mt_m \quad (\text{seconds}) \quad (1)$$

where $t_m = \max[t_1, t_2, \dots, t_n]$

Type 2 Algorithms

For each stage i ($i = 1, \dots, n$) of the pipeline we introduce the numbers x_i and z_i , such that

$$t'_i = x_i t_i$$

and

$$z_i t_i = t_i - t_{i+1} (1 - x_{i+1}) \quad (\text{see Figure 3.8})$$

or

$$\frac{t_i}{t_{i+1}} = \frac{1 - x_{i+1}}{1 - z_i}$$

Figure 3.8 Definition of x_i , z_i and d_i

In other words $z_{i-1}t_{i-1}$ is the time between the start of execution of operations on level $i-1$ and the start of operations on level i .

The time interval d_{i-1} (see Figure 3.8) is the minimum possible time that must elapse (after the start of operations on stage $i-1$) in order to be possible for some of the processing elements on stage i to start operating, that is, the $(i-1)$ th stage will start producing some results to be passed on to the next stage i after, at least, d_{i-1} sec. If possible, it is desirable to have

$$d_{i-1} = z_{i-1}t_{i-1}$$

in order to minimize the execution time t_e .

The numbers x_i , z_i and d_i depend on the algorithms and the number of operators on each particular stage of the pipeline, and they can only be determined after the structure of the pipeline (namely the number of operators on each stage) has been determined. This will become clear in Section D.

For this case the execution time t_e is

$$t_e = z_1t_1 + z_2t_2 + \dots + z_{m-1}t_{m-1} + Mt_m + x_{m+1}t_{m+1} + \dots + x_nt_n$$

or

$$t_e = \sum_{i=1}^{m-1} z_i t_i + M t_m + \sum_{i=m+1}^n x_i t_i \quad (\text{sec}) \quad (2)$$

It is intuitively expected that, under continuous operation ($M \rightarrow \infty$), the efficiency E of the systems considered here will always be less or equal to one. A mathematical proof of this is given in the Appendix A. This leads us to the following definition:

Definition: A processor is said to be matched to an algorithm when the efficiency E of the processor continuously executing ($M \rightarrow \infty$) the algorithm is equal to one.

The following theorem provides the necessary and sufficient conditions for the processor-algorithm matching.

Theorem 1: The efficiency of a processor (with the structure considered here) continuously executing ($M \rightarrow \infty$) an n -level algorithm is one iff

$$\frac{N_1}{rp_1} = \frac{N_2}{rp_2} = \dots = \frac{N_n}{rp_n} = \frac{\sigma}{r \sum_{i=1}^n p_i} = t$$

where t is an integer number.

Proof: Consider an n-level Type 2 algorithm. Under continuous operation the efficiency E of the system becomes (see Appendix A, Equation A.1).

$$\lim_{M \rightarrow \infty} E = \frac{\sigma}{r \left[\frac{N_m}{rp_m} \right] \sum_i p_i} \quad (1)$$

a) Assume that $\lim E=1$, then (1) yields

$$\left[\frac{N_m}{rp_m} \right] = \frac{\sigma}{r \sum_i p_i} \quad (2)$$

assuming that $t_m > t_i$ ($t_m = \max [t_1, t_2, \dots, t_m]$), $\forall i$,

$$\left[\frac{N_m}{rp_m} \right] > \left[\frac{N_i}{rp_i} \right] \geq \frac{N_i}{rp_i}$$

which implies that

$$\left[\frac{N_m}{rp_m} \right] > \frac{\sum_i N_i}{r \sum_i p_i} \quad (3)$$

*Note: If $x > \frac{A_i}{B_i}$, $\forall i$ in $(1, 2, \dots, n)$ then

$$x > \frac{A_1 + \dots + A_n}{B_1 + \dots + B_n}, \quad \forall A_i, B_i > 0$$

but equation (2) is contradictory to equation (3) which means that the hypothesis of $t_m > t_i$ is not valid, therefore

$$t_m = t_i = t \quad (C.1)$$

and

$$t_i = \left[\frac{N_i}{rp_i} \right] = \frac{\sigma}{r \sum_{i=1}^n p_i} \quad (C.2)$$

Now, assume that

$$\left[\frac{N_i}{rp_i} \right] > \frac{N_i}{rp_i}$$

which means

$$t > \frac{N_i}{rp_i}$$

or (see last footnote)

$$t > \frac{\sum_{i=1}^m N_i}{\sum_{i=1}^n p_i}$$

which is contradictory to (C.2), therefore

$$t = \left\lceil \frac{N_i}{rp_i} \right\rceil = \frac{N_i}{rp_i} = \text{integer} \quad (\text{C.3})$$

Equations (C.2) and (C.3) provide the necessary condition of this theorem.

b) For the proof of the sufficient condition of this theorem, see Appendix B.

Q.E.D.

In the analysis so far, there has been no distinction between processing elements. Processing elements assigned to a segment of the pipeline were assumed to be homogeneous and capable of executing any operation of the corresponding level of the algorithm. However, it is often desirable to use single function processing elements (e.g., multipliers, adders, etc.). The optimum distribution of single function processing

elements (that consist of a pipeline segment) among the operations of the level of the algorithm they have been assigned to, is provided by the following theorem.

Theorem 2: Let P_i be the number of single function processing elements that can execute operations of the i th kind, only. If F_i independent operations of the i th kind are required, and $i=1,2,\dots,k$, then the efficiency of a processor consisting of

$$\sum_{i=1}^k P_i$$

processing elements and executing the set of these operations will be one iff

$$\frac{F_1}{rP_1} = \frac{F_2}{rP_2} = \dots = \frac{F_k}{rP_k} = t$$

where t is the time required for the execution of the

$$\sum_{i=1}^k F_i$$

functions, and r is the execution rate of a processing element in ops.

Proof: See Appendix C.

The execution time of a processor matched to a type 1 or 2 n-level algorithm can now be calculated as follows:

Type 1 algorithms:

Equation (1) yields

$$t_e = (n-1)t + Mt \quad (\text{sec}) \quad (3)$$

where t is as in Theorem 1.

Type 2 algorithms:

Since

$$t = t_i \quad (\text{i.e. } t \text{ is equal to } t_i \forall i)$$

we get

$$x_{i+1} = z_i$$

then equation (2) becomes

$$t_e = \left(\sum_{i=2}^n x_i + M \right) t \quad (\text{sec}) \quad (4)$$

and, again t is provided by Theorem 1.

D. Applications

FFT-Processor Design

All the results presented in the previous section will be applied now to the design of an FFT processor.

As it is well known the N-point FFT is an algorithm, in which the original N-point sequence is decomposed into shorter sequence DFTs, thereby reducing the number of computations [15]. For a radix-2 the fundamental two-point transformation is known as a butterfly. The butterfly for decimation in time (DIT) combines two complex input points A and B to give the two complex output points X and Y the operation [15]

$$X = A + W_N^k B$$

$$Y = A - W_N^k B$$

where W_N^k are the complex coefficients.

Figure 3.9(a) shows the flow graph for the DIT butterfly and Figure 3.9(b) depicts the corresponding hardware algorithm (including arithmetic and data transfer (DT) operations) of the same operation.

(a)

(b)

**Figure 3.9 : (a) Decimation in Time Butterfly Flow Graph
(b) The Corresponding Hardware Algorithm**

The butterfly algorithm for decimation in frequency (DIF) is given by

$$\begin{aligned} X &= A+B \\ Y &= (A-B)W_N^k \end{aligned}$$

The flow graph for the DIF butterfly algorithm is shown in Figure 3.10(a). The corresponding hardware algorithm is shown in Figure 3.10(b).

An N -point FFT algorithm requires $\log_2 N$ stages (algorithm levels) with $N/2$ butterfly operations on each stage of the algorithm.

The design of a pipelined FFT processor mainly consists of two parts, namely, the design of a butterfly processor (butterfly module), and the design of the N -point FFT processor that consists of several butterfly modules.

Butterfly Module Design

Both butterfly operations are Type 2, 5-level algorithms (see Figures 3.9(b) and 3.10(b)).

(a)

(b)

Figure 3.10(a) Decimation in Frequency Butterfly Flow Graph
 (b) The Corresponding Hardware Algorithm

a) DIT Module

In the DIT algorithm there are 4 basic operations per level. The 3rd level consists of non-homogeneous operators, namely, adders and DT operators; therefore we use the result of Theorem 2 to determine the proper allocation of the processing elements on this stage of the pipeline, along with the results of Theorem 1.

For simplicity let us assume that the execution rate of each PE is one basic operation per processor cycle, and a processor cycle is equal to the time required by a PE to execute a basic operation (i.e., addition, multiplication, or a data transfer). Then by using Theorem 1 we get the following condition:

$$\frac{4}{P_1} = \frac{4}{P_2} = \frac{4}{P_3} = \frac{4}{P_4} = \frac{4}{P_5} \quad (a)$$

which yields $p_1 = p_2 = p_3 = p_4 = p_5 = 2$, $t=2$ cycles

$$\text{or } p_1 = p_2 = p_3 = p_4 = p_5 = 4, t=1 \text{ cycle}$$

plus the condition for the non-homogeneous 3rd level, as provided by Theorem 2, i.e.,

$$\frac{2}{P_{DT}} = \frac{2}{P_A} = t \quad (b)$$

where p_{DT} and p_A are the required number of DT operators and adders respectively.

Obviously, the t of condition (b) should be the same as the t of condition (a), thus

$$t = 2 \text{ cycles, } p_{DT} = p_A = 1$$

or

$$t = 1 \text{ cycle, } p_{DT} = p_A = 2$$

therefore, for the case of $t = 2$ cycles, the pipelined processor structure will be as follows:

- 1st stage : 2 DT operators
- 2nd stage : 2 multipliers
- 3rd stage : 1 DT operator and 1 adder
- 4th stage : 2 adders
- 5th stage : 2 DT operators

The performance of this butterfly module can be calculated now as follows:

Equation (4) of Section C yields for $t = 2$ cycles

$$t_e = 2 \sum_{i=2}^n x_i + 2M \quad (\text{cycles})$$

and for this case (see Figure 3.9(b) and Figure 3.11(a))

$$x_2=1, x_3= x_4= x_5= 1/2$$

(a)

(b)

Figure 3.11: Execution Timing for the Butterfly Processors

(a) DIT

(b) DIF

thus

$$t_e = 5 + 2M \text{ (cycles)}$$

and if $M \rightarrow \infty$ the execution rate becomes 2 butterflies/cycle.

b) DIF Module

In the DIF algorithm we have the following number of operations per level (see Figure 3.10(b))

$$N_1=4, N_2=N_3=6, N_4=N_5=2$$

Levels 2 and 3 are non-homogeneous consisting of 4 additions - 2 DT operations and 2 DT operations - 4 multiplications respectively.

Working as in the DIT module case, we get the following pipeline structure ($t = 2$ cycles).

- 1st stage : 2 DT operators
- 2nd stage : 2 adders, 1 DT operator
- 3rd stage : 2 multipliers, 1 DT operator
- 4th stage : 1 adder
- 5th stage : 1 DT operator

In this case (see Figure 3.10(b) and Figure 3.11(b))

$$x_2 = x_3 = x_5 = 1/2 \text{ and } x_4 = 1$$

Thus

$$t_e = 5 + 2M \text{ (cycles)}$$

which is the same (as expected) as in the DIT module case.

N-Point FFT Design

There are several forms of FFT algorithms. In this example we will deal with the design of an N-point radix 2, in place FFT processor. In this algorithm, given that N is a power of 2, number N is decomposed into $2 \times N/2$ and then further $N/2$ is decomposed into $2 \times N/4$, etc. [15]. There are $N/2$ DIF butterflies on each one of $\log_2 N$ levels of the algorithm. Figure 3.12 shows an 8-point, radix 2 in place FFT algorithm.

Let us denote t_k the time required to compute k butterflies. Then the timing diagram for the execution of an N-point- DIF, FFT, by a pipelined processor consisting of $\log_2 N$ pipeline stages, will be as in Figure 3.13. The execution time for M executions of the N-point algorithm will then be

Figure 3.12: 8-point FFT Algorithm

Figure 3.13: Execution Timing for the N -point FFT Processor

$$t_e = \sum_{i=2}^{\log_2 N} t_{(N/2^i)+1} + M t_{N/2}$$

Each stage of the pipeline may consist of one or up to $N/2$ DIF butterfly modules. The simplest processor of this kind will consist of one butterfly module per stage. The performance of such a processor will be as follows:

$$t_e = \sum_{i=2}^{\log_2 N} (5 + 2((N/2^i) + 1)) + (5 + 2(N/2))M \text{ (cycles)}$$

assuming that each butterfly module has the structure of the DIF module as determined in the last example. The execution rate of such a processor, when $M \rightarrow \infty$, will be

$$5 + N \text{ (cycles)/N-point FFT}$$

where, "one cycle" = $\max(t_m, t_a, t_{DT})$ and t_m = multiplication time, t_a = addition time, t_{DT} = 1 byte data transfer time. Obviously the optimum case will be when $t_m = t_a = t_{DT}$ (otherwise some computing power will be wasted).

The latency of the structures presented in this section (time between the input of the first data sample and the output of the first result) can be calculated by

setting $M=1$ in the equations for the execution time. In the case of the N -point FFT processor, the latency is approximately proportional to

$$\log_2 N$$

and inversely proportional to the number of butterfly modules used on each segment of the pipeline structure.

E. An Optimization Problem

In this section we shall demonstrate how to use the conditions of Theorems 1 and 2 to formulate an optimization problem for minimizing a cost function of a pipelined system.

In the 6th assumption of Section B, it was assumed that the execution rate of all PEs was the same and equal to r ops. However, it is possible to have PEs with different execution rates. In such a case, a designer would be able to vary the number of PEs required on each stage of the pipeline according to their speed. It could be possible, for example, to use half the number of required multipliers by using multipliers with half multiplication time. However, without cost considerations there will be almost an infinite number of combinations between the number of PEs and their speed

that will satisfy the conditions of Theorems 1 and 2. Let us assume that there is a known relation between the cost of a set of processing elements (e.g., multipliers) and their execution rate, that is, for a particular multiplier i with execution rate r_i , the cost will be

$$c_i = f_m(r_i)$$

where $f_m(x)$ is a known cost-speed function for multipliers. Let us assume that similar functions there exist for adders, DT operators, etc. Then the total cost of a system consisting of x multipliers, y adders and z DT operators will be

$$C = \left(\sum_{i=1}^x f_m(r_{mi}) + \sum_{i=1}^y f_a(r_{ai}) + \sum_{i=1}^z f_d(r_{di}) \right) k$$

where r_{mi} , r_{ai} , r_{di} is the execution rate of multiplier i , adder i , or DT i respectively and k can be a function depending on the total number of PEs consisting of the system (complexity function).

Assuming different execution rates for each PE, the condition of Theorem 1 becomes

$$\frac{N_1}{P_1 \sum_{j=1}^n r_{1j}} = \frac{N_2}{P_2 \sum_{j=1}^n r_{2j}} = \dots = \frac{N_n}{P_n \sum_{j=1}^n r_{nj}} = t$$

where p_{1i} is the number of homogeneous PE consisting of the stage i . A similar relation can be obtained for Theorem 2. Now, the following optimization problem can be formulated:

minimize C

where

$$C = k \left(\sum_{l=1}^x \sum_{i=1}^n p_{li} \right) \left[\sum_{l=1}^x \sum_{i=1}^n \sum_{j=1}^{p_{li}} f_l(r_{lij}) \right]$$

subject to

$$\begin{array}{ccccccc} N_1 & & & N_2 & & & N_n \\ \hline p_{11} & & & p_{yz} & & p_{oz} & p_{xn} \\ \sum_{j=1} r_{11j} & & & \sum_{j=1} r_{yzj} + \sum_{j=1} r_{ozj} & & & \sum_{j=1} r_{xnj} \end{array} = \dots = t$$

p_{li} = integers, $p_{li} \geq 0$, $r_{lij} \geq 0$, and

if $p_{li} = 0$, then $1 \leq \sum_{j=1}^w p_{ly} \leq N_y$

(this means that for a particular level we cannot have more PEs than operations). Unknowns are p_{li} and $r_{lij} \forall l, i, j$ and t is a given specification that determines the performance of the system (see Theorems 1 and 2).

Below we summarize the definition of symbols used above:

C = total cost of the system

$k(h)$ = the complexity function of the system, where h is the total number of all PEs in the system

n = number of pipeline stages (or algorithm levels)

x = number of different kinds of processing elements

$f_l(r)$ = cost-speed function for PEs of the l th kind

r_{lij} = execution rate of the j th PE of the l th kind or the j th stage of the pipeline

p_{li} = number of processing elements of the l th kind on the i th stage of the pipeline

For VLSI systems the cost-speed and complexity functions could be determined by taking into account the silicon area occupied by the system, the technology, as well as the architecture of the system. However, more research has to be done on this subject.

3.3.3 Conclusions

A method of optimizing processors to execute specific DSP algorithms has been outlined in this section. The results provide pipelined processing structures, that are highly modular (therefore well suited for VLSI implementation) and characterized by minimal

processing and data flow overhead. This is achieved by allocating the various operations of a DSP algorithm to processing elements consisting of the segments of the pipeline in such a way that all basic components are concurrently active throughout the operation. With the recent advent of VLSI, these complex but high performance structures can be built with reasonable size and cost.

Finally, the optimization problem outlined in Section E could provide the means for determining (or at least predicting) the optimum (for a particular algorithm) system, under cost and performance considerations, in the early stages of the design.

3.4 Summary

In this chapter, the basis has been established for both judging the appropriateness of a task assigned to a processor, and for actually synthesizing a processor for a particular task. Architectural efficiency formed the concept on which both the analysis and the synthesis algorithms have been built.

The main issue was presented in Section 3.3, where, the problem of optimally allocating DSP algorithms to pipelined architectures of processing elements, has been given a definitive solution.

Another aspect in the design of DSP systems will be examined in the following chapter which deals with the trade-offs associated with the processing power of multiprocessing VLSI systems and certain constraints imposed by pin limitations.

3.5 References

- [1] R.R. Muntz, E.G. Coffman, "Optimal Preemptive Scheduling on Two-Processor Systems," IEEE Trans. Computers, Vol. C-18, No. 11, pp 1014-1020, Nov. 1969.
- [2] C.V. Ramamoorthy, K.M. Chandy, and M.J. Gonzalez, "Optimal Scheduling Strategies in a Multiprocessor System," IEEE Trans. Computers, Vol. C-21, No. 2, pp 137-146, Feb. 1972.
- [3] E.B. Fernandez, B. Bussel, "Bounds on the Number of Processors and Time for Multiprocessor Optimal Schedules," IEEE Trans. Computers, Vol. C-22, No. 8, pp 745-751, Aug. 1973.
- [4] Zaheer M. Ali, "A High-Speed FFT Processor," IEEE Trans. on Communications, Vol. COM-26, No. 5, pp 690-696, May 1978.
- [5] J.R. Jump and S.R. Ahuja, "Effective Pipelining of Digital Systems," IEEE Trans. Computers, Vol. C-27, No. 9, pp 855-863, Sept. 1978.

- [6] J.M. Glass, P. Kotiveeriah, "The Allocation Problem in Distributed Signal Processing," IEEE Trans. Acoust., Speech, Signal Processing, Vol. ASSP-29, No. 4, pp 817-829, Aug. 1981.
- [7] H.T. Kung and R.L. Picard, "Hardware Pipelines for Multi- Dimensional Convolution and Resampling," Proc. 1981 IEEE Computer Society Workshop on Computer Architecture for Pattern Analysis and Image Database Management, Nov. 1981, pp 273-278.
- [8] H.M. Ahmed, J.M. Delosme, and M. Morf, "Highly Concurrent Computing Structures for Matrix Arithmetic and Signal Processing," IEEE Computer, pp 65-82, Jan. 1982.
- [9] H.T. Kung, "Why Systolic Architectures?" IEEE Computer, Vol. 15, No. 1, pp 37-46, Jan. 1982.
- [10] J.S. Thompson and S.K. Tewksbury, "LSI Signal Processor Architecture for Telecommunications Applications," IEEE Trans. Acoust., Speech, Signal Processing," Vol. ASSP-30, No. 4, pp 613-631, Aug. 1982.
- [11] K.O. Siomalas, B.A. Bowen, "Performance of Cross-bar Multiprocessor Systems," IEEE Trans.

- Computers, Vol. C-32, No. 7, pp 689-695, July 1983.
- [12] G. Bongiovanni, "A VLSI Network for Variable Size FFTs," IEEE Trans. Computers, Vol. C-32, No. 8, pp 756-760, Aug. 1983.
- [13] P. Chow, Z.G. Vranesic, and J.L. Yen, "A Pipelined Distributed Arithmetic PFFT Processor," IEEE Trans. Computers, Vol. C-32, No. 12, pp 1128-1136, Dec. 1983.
- [14] E.H. Wold and A.M. Despain, "Pipeline and ParallelPipeline FFT Processors for VLSI Implementation," IEEE Trans. Computers, Vol. C-33, No. 5, pp 414-426, May 1984.
- [15] L.R. Rabiner and B. Gold, "Theory and Applications of Digital Signal Processing," Prentice-Hall 1975.
- [16] S.Y. Kung, "On Supercomputing with Systolic Wavefront Array Processors" Proc. IEEE, Vol. 72 No. 7, pp 867-884, July 1984.

CHAPTER 4. Performance Constraints in VLSI Implementation

4.0 Introduction

The processing structures proposed in Chapter 3 may be implemented in VLSI with reasonable size and cost. There are several problems associated with VLSI implementation of high performance digital systems. Most of them, however, are beyond the scope of this work.

A major implementation constraint of large systems in the VLSI environment, is the number of pins available on a chip. This problem cannot be overlooked even in the early stages of the design of any high performance system. Limitations on the number of pins available on chips may affect the performance, the efficiency, and, possibly, the architecture of the systems to be implemented. This chapter deals with this problem, and some definitive answers are given.

4.1 Processing Element Multiplicity and Pin Limitations

4.1.1 Introduction

One of the problems encountered in the design of pipelined or parallel-pipelined digital signal processing

(DSP) VLSI systems is the handling of the large number of data bus lines that interconnect the various processing elements of a system [5], [6]. Since data and results must be received or be sent in parallel, multiple bus lines and I/O ports are needed to avoid conflicts or bottlenecks. Generally, in the design of VLSI systems, it is required either to minimize the number of bus lines and I/O ports by properly interconnecting and synchronizing the various operations of the system, [2], [3], [4], or, when limitations are imposed on the number of the available bus lines or I/O ports, to maximize the performance by increasing the number of parallel processing elements to a maximum such that no conflicts or bottlenecks can occur.

In this chapter we investigate this problem, namely, we examine ways to increase the performance of DSP multiprocessing systems to a maximum under the constraint of a limited number of I/O ports (limiting the number of I/O ports is always a necessary action in the design of any VLSI system, since there is always a limitation on the number of pins that a single chip can bear).

In Section 4.1.2, we examine the performance of VLSI systems that use single input and single output ports. The problem is extended in Section 4.1.3, to VLSI chips

with a single I/O port that performs time multiplexed I/O operations.

4.1.2 Performance of VLSI Systems with Single-Input and Single-Output Ports

The block diagram of a VLSI chip consisting of multiple processing elements (PEs) is shown in Figure 4.1. Without loss of generality, we assume the following:

1. We consider VLSI systems consisting of several identical PEs operating continuously (a typical case of DSP components) in parallel.
2. Each PE has its own input and output registers.
3. All PEs share a single input and a single output port through which they can communicate with the external world.
4. The input or the output ports can serve one PE at a time.
5. Let t_{tr} be the time required by a data transfer device (DT) to fetch the required data, t_p be the time required by a PE to process the

Figure 4.1 : Block Diagram of a Single-Input Single-Output Port VLSI System, Consisting of Identical PEs

incoming data, and t_d the time the data must stay in the I/O port before it is transferred to the input registers of a PE or of an external device. Thus the time t_d is limiting the maximum input and output rate to $1/t_d$ data transfers per second. It is assumed that t_d is much smaller than both t_{tr} and t_p .

The following two cases can occur.

a) $t_{tr} < t_p$

Figure 4.2 shows the timing for this case. The vertical axis indicates the particular PE or DT that is examined. For example the data transfer operator DT1 supplies the first data set to PE1 and, since $t_{tr} < t_p$, at a time equal to $2t_{tr}$ it supplies with data a second processing element PE2, while the first PE1 is still processing the previous data. The dashed lines indicate the execution timing of another DT operator (i.e., DT operator No. 2) and another set of processing elements which is supplied with data by the operator DT2. As long as the time between two successive I/O operations is greater or equal to t_d , many of these data transfer operators and processing element sets can coexist. In the next section we shall investigate this

Figure 4.2: Timing Diagram for the Case of $t_{tr} < t_p$

problem. For the moment we assume that there is no I/O port conflict.

As can be seen in Figure 4.2, since the rate of the incoming data is greater than the processing rate of a PE, a set of more than one PE is required to achieve the maximum execution rate of $1/t_{tr}$ ops (operations per second).

Let t_e be the execution time of N basic operations performed by a system consisting of m_p processing elements and supplied by m_d data transfer operators. Assuming that a basic operation is performed in t_p sec by a single PE, then

$$t_e = t_{tr}N + t_p + t \text{ (sec)}$$

where t is a constant comparable to t_p . Now let U be the utilization of the set of the m_p PEs, then

$$U = \left(\frac{\frac{N}{m_p} t_p m_d}{t_e} \right) N \rightarrow \infty$$

since under continuous operation $N \rightarrow \infty$, thus

$$U = \frac{t_p m_d}{t_{tr} m_p}$$

for $U = 1$

*Note : U is defined as $U = (\text{min time required to execute } N \text{ operations by } m_p \text{ operators without port constraints}) / (\text{actual execution time})$,
 or, $\frac{(\text{number of input data}) \times (t_p / m_p)}{t_e} = U$

$$m_p = \left\lceil \frac{t_p}{t_{tr}} \right\rceil m_d \quad (2.1)$$

Note that for the moment we assume that m_d is such that there is no port contention. Certainly there is an upper bound on m_d beyond which there will be conflicts. This bound will be calculated later on (Section 4.1.3.4).

Now, let R be the execution rate of the system under continuous operation, then

$$R = \left(\frac{Nm_d}{t_e} \right)_{N \rightarrow \infty} = \frac{m_d}{t_{tr}} \text{ ops}$$

or

$$m_d = R t_{tr} \quad (2.2)$$

Equations (2.1) and (2.2) provide the required number of PEs and DTs to achieve the required execution rate R .

*Note : $m_p \leq \left\lceil \frac{t_p}{t_{tr}} \right\rceil m_d$

2 2
OF / DE

$$b) \quad t_{tr} \geq t_p$$

The timing for this case is shown in Figure 4.3. By following the same procedure as in case (a), we can calculate the following:

$$U = \left(\frac{N \frac{t_{tr}}{m_d}}{t_e / m_p} \right)_{N \rightarrow \infty} = \frac{m_p t_{tr}}{t_p m_d}$$

for 100% utilization ($U=1$) the required number of DT operators is

$$m_d = \left\lceil \frac{t_{tr}}{t_p} \right\rceil m_p \quad (2.3)$$

the execution rate in this case is

$$R = \left(\frac{m_p N}{t_e} \right)_{N \rightarrow \infty} = \frac{m_p}{t_p} \text{ ops}$$

*Note : $m_d \leq \left\lceil \frac{t_{tr}}{t_p} \right\rceil m_p$

Figure 4.3 : Timing Diagram for the Case of $t_{tr} > t_p$

or

$$m_p = R t_p \quad (2.4)$$

In this case equations (2.3) and (2.4) provide the required number of DTs and PEs to achieve a given execution rate R .

4.1.3 VLSI Systems with a Time Multiplexed I/O Port

4.1.3.1 Introduction

Time multiplexing of I/O ports is a common approach in the design of a VLSI system. The main reason for this is the technological limitations imposed on the pin capacity of a VLSI chip. In DSP systems, where the word length is essential to the computational accuracy of the system, this problem becomes particularly important. For example, the typical word length of 32 bits requires 64 pins just for data if different ports are used for input and output operations. The number of data pins can be reduced to a half by using the same port both for input and output data; however, this could result in a significant performance degradation, particularly in the case of the pipelined multiprocessing structures considered in this work, where, many sets of data must move in and/or out of the chip in parallel. So far,

with the exception of some experimental results [1], there has been no theoretical work in the literature dealing with this problem.

In this section, we shall examine the problem of maximizing the multiplicity of processing elements that can operate in parallel in a VLSI system, where time-multiplexed I/O ports are being utilized.

4.1.3.2 Simple Models

Consider a VLSI system consisting of several identical processing elements. The following assumptions are made concerning the operation of the system.

1. There exists one I/O port that can be used either as input or as output port. Only one I/O operation at a time is possible. The PEs share the I/O port to input, or, output data into, or, out of the system (chip).
2. Let t_d be the I/O port delay. It is the minimum time required between two successive I/O operations. Let t_x be the delay between two successive operations of the same PE; t_x can be the restart time of a PE or a delay introduced for synchronization purposes. It must be $t_x \geq t_d^*$. Finally, t_p is the

*Note : $t_x = t_{re} + t_d$, where t_{re} is the actual restart time ($t_{re} \geq 0$)

processing time as defined in Section 4.1.2. It is assumed that $t_p \geq t_x$.

The following theorem provides the upper bound on the number of the PEs that can operate concurrently and use the same I/O port without conflict.

Theorem 1: The maximum number m of the PEs that can operate in parallel and share a single I/O port for data movements without conflict, is

$$m = m_1 + m_2 + m_3$$

where

$$m_1 = \left\lfloor \frac{t_x}{t_d} \right\rfloor$$

$$m_2 = nm_1$$

where

$$n = \left\lfloor \frac{t_p - m_1 t_d}{f} \right\rfloor$$

$$f = t_x + m_1 t_d$$

and

$$m_3 = k_3 u(k_3), \quad k_3 = \left\lfloor \frac{t_p - f(n+1) - t_d}{t_d} \right\rfloor$$

where $u(k_3)$ is the step function of k_3 .

Proof: Due to the time interval t_x , m_1 processing elements can operate in parallel (with a delay of t_d sec between two PES), without I/O port conflict (see Figure 4.4). Thus

$$m_1 = \left\lfloor \frac{t_x}{t_d} \right\rfloor$$

The m_1 PES will be occupying the I/O port in the time interval between t_p and,

$$t_p + (t_x + (m_1 - 1)t_d)$$

Thus, an additional processing element to m_1 elements can start operating at a time equal to

$$t_p + f - t_p$$

where

$$f = (t_x + (m_1 - 1)t_d) + t_d$$

or

$$f = t_x + m_1 t_d$$

Figure 4.4 : Timing Diagram Used for the Proof of Theorem 1

Now, if n is the maximum number of f -time intervals in t_p ,

$$m_2 = nm_1$$

additional processing elements can be used without I/O conflict (i.e., conflict with the outputs of the 1st set of m_1 elements), if

$$t_p - nf \geq (m_1 - 1)t_d + t_d$$

or

$$n^* = \left\lfloor \frac{t_p - m_1 t_d}{f} \right\rfloor$$

and

$$m_2 = nm_1$$

In case the ratio $(t_p - m_1 t_d)/f$ is not an integer number, then there could be space for $m_3 \leq m_2 - 1$ additional PEs; thus, if the remaining time between the input of the last element of the 1st set of m_1 elements and t_p is

$$t_p - f(n+1) - t_d$$

m_3 will be

$$m_3 = k_3 u(k_3), \quad k_3 = \left\lfloor \frac{t_p - f(n+1) - t_d}{t_d} \right\rfloor$$

where

$$u(k_3)=0 \text{ when } k_3 < 0$$

$$\text{and } u(k_3)=1 \text{ when } k_3 \geq 0$$

Q.E.D.

The utilization U and the execution rate R of such a system (i.e., a system consisting of m PEs) under continuous operation can be calculated as follows:

Let t_e be the execution time for N operations and t_{em} be the execution time for Nm operations, then

$$t_{em} = Nt_p + (N-1)t_x + t \text{ (sec)}$$

and

$$t_e = \frac{t_{em}}{m} \text{ (sec)}$$

where t is a constant comparable to t_p

then

$$U = \left(\frac{N \frac{t_p}{m}}{t_e} \right) \quad N \rightarrow \infty$$

or

$$U = \frac{1}{1 + \frac{t_x}{t_p}} \quad (3.1)$$

from equation (2.4) we get the execution rate which in this case becomes

$$R = \frac{m}{t_p + t_x} \quad (3.2)$$

since (Theorem 1)

$$m \leq \frac{t_x}{t_d} + \frac{t_p - t_x}{2t_d}$$

equation (3.2) becomes

$$R \leq \frac{1}{2t_d} \quad (3.3)$$

Thus, $1/2t_d$ is the upper bound of the execution rate of such a system.

4.1.3.3 General Models

In this subsection we shall develop a general model for the determination of the maximum number of PEs that can operate concurrently in a VLSI system that has a single time-multiplexed I/O port.

The same assumptions of Section 4.1.3.2 are applied in this case but with the following changes.

Let t_{op} be the operation time of a PE, then

$$t_{op} = t_i + t_p + t_o$$

where

t_i = time required by a PE to load a set of data into the chip (I/O port will be busy during t_i)

t_p = time required by a PE to process the data. The I/O port will not be used by this particular PE during t_p

t_o = time required by a PE to output the result(s) (I/O port will be busy during t_o , as well)

and t_x is a time as defined in Section 4.1.3.2 (time required to restart the operation of a PE on a new input data set). For the moment we assume that $t_i \geq t_o$ and $t_x < t_p$. The maximum number of PEs that can be concurrently active in such a system is provided by the following theorem.

Theorem 2: The maximum number of PEs (with the above characteristics) that can be concurrently active and share the same I/O port without conflict is:

$$m = m_1 + m_2 + m_3$$

where

$$m_1 = \left\lfloor \frac{t_x}{t_i} \right\rfloor + 1$$

$$m_2 = nm_1$$

$$m_3 = k u(k), \quad k = \left\lfloor \frac{t_i + t_p - (n+1)f}{t_i} \right\rfloor$$

where

$$n = \left\lfloor \frac{t_i + t_p - m_1 t_i}{t_o + t_x + m_1 t_i} \right\rfloor$$

$$f = t_o + t_x + m_1 t_i$$

and $t_i, t_o \neq 0$

Proof:

The proof is similar to the proof of Theorem 1. In this case

$$m_1 = \left\lfloor \frac{t_x}{t_i} \right\rfloor + 1$$

in the time interval t_1+t_p , t_1+t_p+f (see Figure 4.5) the I/O port is occupied by the I/O operations of the m_1 processing elements, where

$$f = t_o + t_x + m_1 t_i$$

Let n be the max number of f -time intervals that can fit in t_{op} , then $m_2 = m_1 n$ additional PEs can be used without conflict if

$$t_{op} - nf \geq t_o + m_1 t_i$$

or

$$n(t_o + t_x + m_1 t_i) = t_1 + t_p - m_1 t_i$$

then

$$n = \left\lfloor \frac{t_1 + t_p - m_1 t_i}{t_o + t_x + m_1 t_i} \right\rfloor$$

and

$$m_2 = nm_1$$

Similarly to the proof of Theorem 1

$$m_3 = k u(k), \quad k = \left\lfloor \frac{t_{op} - ((n+1)f + t_o)}{t_i} \right\rfloor$$

Q.E.D.

Figure 4.5: Timing Diagram Used for the Proof of Theorem 2

Now, in the case $t_0 > t_i$, these results can be extended as in the following corollary.

Corollary 1: If $t_0 > t_i$ then

$$m = m_1 + m_2 + m_3 + m_4$$

where

$$m_1 = \left\lfloor \frac{t_x}{t_0} \right\rfloor + 1$$

$$m_2 = nm_1$$

$$n = \left\lfloor \frac{t_p - \left\lfloor \frac{t_x}{t_0} \right\rfloor}{f} \right\rfloor$$

$$f = \left\lfloor \frac{t_x}{t_0} \right\rfloor + t_i + t_0 + t_x$$

$$m_3 = k_3 u(k_3), \quad k_3 = \left\lfloor \frac{t_{op} - (n+1)f - t_0}{t_0} \right\rfloor$$

and

$$m_4 = u(k)$$

where

$$k = \left\lfloor \frac{t_{op} - (n+1)f - t_0 - m_3 t_0}{t_i} \right\rfloor$$

Proof:

The proof is similar to the proofs of Theorems 1 and 2. In this case the I/O port is occupied by the first m_1 PEs in the time interval

$$t_i + t_p, t_i + t_p + f$$

where

$$f = \frac{t_x}{t_o} + t_i + t_o + t_x$$

and the condition for the additional $m_2 = nm_1$ PEs is

$$t_{op} - nf \geq t_o + \left\lfloor \frac{t_x}{t_o} \right\rfloor + t_i$$

thus

$$n = \left\lfloor \frac{t_p - \left\lfloor \frac{t_x}{t_o} \right\rfloor}{f} \right\rfloor$$

and

$$m_2 = nm_1$$

as in Theorems 1 and 2,

$$m_3 = k_3 u(k_3)$$

and since $t_i < t_o$, there could be space for one additional PE, $m_4 = u(k_4)$.

Q.E.D.

The reader can easily prove now the following two cases of practical interest (these cases refer back to section 4.1.2; the proofs are given in Appendix)

Case 1: $t_i = 0$

In this case

$$m = 1 + \left\lfloor \frac{t_p + t_x}{t_o} \right\rfloor$$

Case 2: $t_o = 0$

In this case

$$m = 1 + \left\lfloor \frac{t_p + t_x}{t_i} \right\rfloor$$

Note that t_{tr} of Section 4.1.2 is equivalent to $t_p + t_x$ and t_d equivalent to t_o or t_i , of the present section.

In both cases it is assumed that all m operators operate continuously. In Case 1, t_o is the time an external device (operator) uses the input port of the chip to load data that have been produced (or fetched) by this operator in time t_p , and m is the maximum number of such operators that can share the input port. Similarly

in Case 2, t_1 is the time that the output port of a processing device (chip) is occupied by an external operator which reads the output data that have been produced by the chip. The external device then will process these data in t_p time.

Some applications of all these results will be presented in the next subsection.

4.1.3.4 Applications

Before we proceed we shall examine the case where $t_x=0$, which is true for most of the applications. At this point we should note that there is a difference between the time t_x used in subsection 4.1.3.2 and the time t_x of subsection 4.1.3.3 which is rather equivalent to t_{re} as defined in a footnote in subsection 4.1.3.2. Hence, we will deal only with the subsection 4.1.3.3 model which is the general case.

For $t_x = 0$, Theorem 2 yields

$$m = 1 + \left\lfloor \frac{t_p}{t_o + t_1} \right\rfloor \quad (3.1)$$

because

$$m_1 = 1$$

$$f = t_o + t_1, n = \left\lfloor \frac{t_p}{t_o + t_1} \right\rfloor, m_2 = nm_1$$

and

$$m_3 = 0 \text{ (see Appendix for the proof)}$$

The execution rate in this case (under continuous operation) is

$$R = \frac{m}{t_{pr}} \text{ (ops)}$$

or

$$R = \frac{1 + \frac{t_p}{t_o + t_i}}{t_{pr}} \text{ (ops)} \quad (3.2)$$

where

$$t_{pr} = t_i + t_p + t_o$$

from equation (3.2) we get the upper bound of R (see Appendix), i.e.,

$$R \leq \frac{1}{t_o + t_i} \text{ (ops)} \quad (3.3)$$

which occurs when $t_p/(t_o + t_i)$ is an integer number other than zero.

EXAMPLE - MAC chips

We now apply these results to the case of the well known TRW Multiplier-Accumulator chips (TDC1008J,

TDC1009J, TDC1010J). The timing diagram for these three devices is shown in Figure 4.6, [7]. The input register set-up time t_s is equivalent to t_i , the multiply-accumulate time t_{ma} plus the output delay time t_d is equivalent to t_p , and t_{do} is the time the data occupy the output pins (in this case it depends on how fast an external device can read the data from the pins, and it is not a specification of the chip), and therefore equivalent to t_o . Let us assume that t_{do} is equal to t_s . Table 1 shows the value of m for each device, as it is provided by Theorem 2.

Table 1

DEVICE	WRD LENGTH (Bits)	t_s ns	t_{ma} ns	t_d ns	t_i ns	t_p ns	t_o ns	m
TDC1008J	8	12	70	32	12	102	12	5
TDC1009J	12	12	95	32	12	127	12	6
TDC1010J	16	12	115	32	12	147	12	7

GENERAL SYSTEM STRUCTURE

The general structure of a system consisting of several processing element chips (PEC) and data transfer operator chips (DTC) is shown in Figure 4.7. It is assumed that all PEC and DTC chips have the processing

Figure 4.6 : Timing Diagram for the MAC-Chips(7)

Figure 4.7 : Structure of a System Consisting of Several PEC and DTC Operators

element multiplicity (m) as it is provided by the theory in subsections 4.1.3.2 and 4.1.3.3. Assuming that all chips operate at the maximum execution rate, then, each PEC is synchronized with the two DTC when

$$R_{iDTC} = R_{PEC} = R_{oDTC}$$

or

$$\frac{1}{t_{oDT}} = \frac{1}{t_o + t_1} = \frac{1}{t_{iDT}}$$

The I/O buffers in front of each PEC should have a few words capacity, and they must be used for synchronization reasons. Note that the times t_{oDT} and t_{iDT} are equivalent to t_d of section 4.1.2.

TWO PORT VLSI SYSTEMS

The results of subsections 4.1.3.2 and 4.1.3.3 can now be used to calculate the upper bounds on m_p and m_d of the two port systems examined in Section 4.1.2

$$(a) \ t_{tr} < t_p$$

The incoming data rate (from a single source) is $1/t_{tr}$, and (see Figure 4.8(b))

(a)

(b)

Figure 4.8: Timing Diagrams

(a) $t_{tr} > t_p$ (b) $t_{tr} < t_p$

$$t_{tr} = t_{dt} + t_d$$

where t_{dt} is the true data transfer time and t_d as defined in assumption 5 of Section 4.1.2

The multiplicity m_p of the processing elements consisting of the system, which are required to assure continuous processing of the data supplied by m_d DT operators, must be

$$m_p = \lceil t_p / t_{tr} \rceil m_d$$

By setting $m = m_d$, $t_x = 0$, $t_p = t_{dt}$ (t_{dt} of a DT operator is equivalent to the processing time of a PE), and $t_o = t_d$ (t_d is the time required by a DT operator to output the results, therefore it is equivalent to t_o), the result of Case 1 (Section 4.1.3.3) becomes

$$m_d = 1 + \lceil t_{dt} / t_d \rceil \quad (3.4)$$

Equation (3.4) provides the maximum number of the DT operators that can use the output port of the processing system without conflicts,

$$m_p = \lceil t_p / t_{tr} \rceil (1 + \lceil t_{dt} / t_d \rceil) \quad (3.5)$$

or (see Appendix)

$$m_p \leq (t_p + t_{tr} - 1) / t_d \quad (3.6)$$

Using equation (3.6), the lower bound on the utilization $\underline{U}$, of the processing elements, can be obtained (see Appendix)

$$\underline{U} = t_p / (t_p + t_{tr} - 1) \quad (3.7)$$

$$(b) \quad t_{tr} \geq t_p$$

In this case, the execution rate is $1/t_p$, and

$$t_p = t_{pr} + t_d \quad (\text{see Figure 4.8(a)})$$

where t_{pr} is the time required by a PE to process the incoming (with rate $1/t_{tr}$) data.

The multiplicity m_d of the DT operators required to assure continuous operation of m_p PEs (in this case the data transfer is the bottleneck in the system) is

$$m_d = \lceil t_{tr} / t_p \rceil m_p$$

From Section 4.1.3.3, Case 2 (in this case $t_i = t_d$), we get

$$m_p = 1 + \lceil t_{pr} / t_d \rceil \quad (3.8)$$

thus

$$m_d = \lceil t_{tr}/t_p \rceil (1 + \lceil t_{pr}/t_r \rceil) \quad (3.9)$$

or (see Appendix)

$$m_d \leq (t_{tr} + t_p - 1)/t_d \quad (3.10)$$

The lower bound $\underline{U}$ on the utilization of the m_d data transfer operators is (see Appendix)

$$\underline{U} = t_{tr}/(t_{tr} + t_p - 1) \quad (3.11)$$

4.2 Summary and Conclusions

The problem of maximizing the performance of single-chip DSP VLSI systems (or other similar VLSI systems), by using as many concurrently operating processing elements as possible, has been investigated in this chapter.

Limits imposed on the number of concurrently operating processing elements that can be put onto a single chip are due to the limited number of I/O pins that a chip package can bear. Although bottlenecks caused by the limited number of I/O ports can result in a significant performance degradation, it has been shown

that it is still possible to integrate into a single chip a relatively large number of parallel operating processing elements which can share a single I/O port without conflicts. Thus, execution rates beyond the speeds imposed by technological limitations could be achieved. A similar analysis can also be applied to the case of the limited number of control lines of a multiprocessing element chip.

The question of whether it is possible to implement a VLSI chip that will use the predicted processing element multiplicity is still open. However, the presented results provide upper bounds and a methodology which, as the VLSI technology progresses, could prove to be very useful in the design of any high performance system.

4.3 References

- [1] E. Bloch and D. Galaze, "Component Progress: Its effect on high speed computer architecture and machine organization", In High Speed Computer and Algorithm Organization, D. Kuck, D. Laurie, A. Sameh, Eds. New York, Academic Press 1977.
- [2] H. Siegel et al., "A Survey of Interconnection Methods for Reconfigurable Parallel Processing Systems", Proc. 1979 NCC, AFIPS, vol. 48, June 1979, pp 529-542.
- [3] M. A. Franklin, D. F. Wann, and W. J. Thomas, "Pin limitations and Partitioning of VLSI Interconnection Networks", IEEE Trans. Computers, vol. C-31, No 11, pp1109-1116, Nov. 1982.
- [4] T. Lang, M Valero, and M.A. Fiol, "Reduction of Connections for Multibus Organization", IEEE Trans. Computers, vol. C-32, No. 8, pp707-716, Aug. 1983.
- [5] P. Chow, Z. G., Vranesic, and J. L. Yen, "A Pipelined Distributed Arithmetic PFFT Processor", IEEE Trans. Computers, vol., C-32, No. 12, pp 1128-1136, Dec. 1983.
- [6] E. H. Wold, and A. M. Despain, "Pipeline and Parallel-Pipeline FFT Processors for VLSI

Implementations", Trans. Computers, vol. C-33, No 5, pp 414-426, May 1984.

- [7] TRW LSI Products 1979, Multiplier-Accumulators Parallel 8, 12, or 16 bits.

CHAPTER 5. Conclusions

The principal focus of this work has been the matching of algorithms and architectures to achieve highly efficient and very high performance Digital Signal Processing systems specialized to execute specific algorithms. This problem requires an understanding of both the structure of the algorithm and an appreciation of the architectural and implementation constraints. It has been shown that there is an optimum partition and allocation of an algorithm, such that the performance and the efficiency of the resulting processor is maximized.

It has also been shown that a number of independent processing elements, when synchronized properly, can share the same I/O port without conflicts, and still achieve a high degree of utilization.

The novelties of the present work also include:

- Consideration of data transfers in the modelling. This problem is usually ignored or treated after the fact.

-Linear increase of performance. Theoretically, any given performance requirement can be met (at the maximum efficiency), by employing the appropriate number of processing elements, as is shown in Chapter 3.

-Consideration of pinout limitations. The results of this study (Chapter 4), provide the upper bound on performance and on the number of processing elements that can be realized in a system constrained by an I/O port bottleneck.

In summary, the structures of the proposed DSP systems are characterized by the following:

(a) Highly efficient structure. The partitioning of algorithm operations and the allocation to processing elements is such that all the basic components of the system will be concurrently active throughout the operation. Therefore, processing and data flow overhead is minimal.

(b) Modular structure suitable for VLSI implementation. With the recent advent of VLSI, many processing elements can be realized on a single chip. Design emphasis has shifted from minimizing complexity to partitioning processor functions on circuits and managing communications between partitions. The design philosophy

throughout in this work has been consistent with this concept.

The results of the present work can be the starting point for further research, which may include one of the following topics:

(a) Systolic Array Architectures: The results of Chapters 3 and 4 could be extended to the design of the recently introduced systolic architectures. For example, it should be possible to develop a similar methodology to determine the optimum structure of a systolic array required for a particular application.

(b) VLSI Multiprocessing Systems: An analysis similar to that of Chapter 4 can be applied to the case of the limited number of control lines in a multiprocessing VLSI system.

(c) Develop a method to determine (or approximate) the relation between performance and cost of certain processing elements: As we have already mentioned in Section E of Chapter 3, these results are needed to complete the processor optimization problem.

Finally, the results of this work provide the theoretical basis for finding the optimal partition and allocation of DSP algorithms to processing elements, and

for partitioning the PEs onto either chips or boards to minimize the effects of I/O port bottlenecks. These results can be used by the new technology, namely, SaAs, WSI (Wafer Scale Integration), and CAD (silicon compilers), to produce very efficient and very high performance DSP systems.

APPENDIX

Chapter 3

$$A) \lim_{M \rightarrow \infty} E < 1$$

Proof: The efficiency E has been defined in Section 3.3.2,C as

$$E = R / (\text{Total Computing Power})$$

or

$$E = R / r \sum_{i=1}^m p_i$$

where

$$R = \sigma M / t_e \text{ (ops)}$$

thus

$$E = \frac{M \sigma}{r t_e \sum_{i=1}^m p_i}$$

or by using equation (3)

$$E = \frac{M \sigma}{r \left(\sum_{i=1}^{m-1} z_i \left[\frac{N_i}{r p_i} \right] + M \left[\frac{N_m}{r p_m} \right] + \sum_{i=m+1}^n x_i \left[\frac{N_i}{r p_i} \right] \right) \sum_{i=1}^n p_i}$$

when $M \rightarrow \infty$, E becomes

$$\lim_{M \rightarrow \infty} E = \frac{\sigma}{r \left[\frac{N_m}{r p_m} \right] \sum_{i=1}^n p_i} \quad (A.1)$$

but since

$$\left[\frac{N_m}{rp_m} \right] = t_m \geq t_j = \left[\frac{N_j}{rp_j} \right], \forall j \neq m$$

equation (A.1) becomes

$$\lim_{M \rightarrow \infty} E_j \leq \frac{\sigma}{\sum_i p_i} \frac{p_j}{N_j}$$

or

$$\lim_{M \rightarrow \infty} E_j \leq \frac{\sigma}{\sum_{i=1}^m p_i} \cdot \frac{\sum_{j=1}^n p_j}{\sum_{j=1}^m N_j} = 1$$

Q.E.D.

*Note that if $x \leq \frac{A_j}{B_j}$ then $x \leq \frac{\sum_j A_j}{\sum_j B_j}$, $\forall A_j, B_j \geq 0$

B) Proof of the sufficient condition of Theorem 1. We will show that if t is an integer number and

$$t = \frac{N_i}{rp_i} \quad \text{then } \lim_{M \rightarrow \infty} E = 1$$

Since t is integer and

$$\frac{N_i}{rp_i} = \frac{\sigma}{r \sum_i p_i}$$

equation (A.1) becomes

$$\lim_{M \rightarrow \infty} E = 1$$

$M \rightarrow \infty$

Q.E.D.

C) Proof of Theorem 2

The execution time t must be

$$t = \max \left(\left\lceil \frac{F_1}{rp_1} \right\rceil, \dots, \left\lceil \frac{F_k}{rp_k} \right\rceil \right)$$

$$\text{let } t = \left\lceil \frac{F_m}{rp_m} \right\rceil$$

then the efficiency of the system will be

$$E = \frac{\sum_{i=1}^k F_i / t}{r \sum_{i=1}^k p_i}$$

Now assume that $E = 1$ then

$$\left[\frac{F_m}{rP_m} \right] = \frac{\sum_{i=1}^k F_i}{rLP_i}$$

the rest of the proof now is similar to that of Theorem 1.

APPENDIX I

21:22 MAR 28 '83

```

1 - 1.000 C ** DSP CONTROL SIMULATION PROGRAM **
2 - 2.000 C THIS PROGRAM IS WRITTEN IN FORTRAN 77 VERSION B04
3 - 3.000 C *** NOTES ***
4 - 4.000 C 1) THE PROGRAM REQUIRES 57K MAIN MEMORY
5 - 5.000 C AND IT CAN BE USED FOR UP TO 65 EXECUTIONS OF
6 - 6.000 C THE SAME ALGORITHM.
7 - 7.000 C 2) THE PROGRAM ASSUMES ALGORITHM MATRIX WITH TWO
8 - 8.000 C DIGIT ELEMENTS, THUS FOR EXAMPLE A 5 MUST BE EN-
9 - 9.000 C TERED AS 05.
10 - 10.000 C 3) THE ALLOCATION PRINTOUT MUST BE USED FOR LI-
11 - 11.000 C MITED No OF EXECUTIONS (UP TO 100 CYCLES).
12 - 12.000 C
13 - 13.000 C
14 - 14.000 C INTEGER MUL(65,350),X,S,T,P
15 - 15.000 C INTEGER AD(65,350),ALG(20,10),AUX(10,400)
16 - 16.000 C DO 6 I=1,65
17 - 17.000 C DO 69 J=1,350
18 - 18.000 C 69 MUL(I,J)=0
19 - 19.000 C 6 CONTINUE
20 - 20.000 1000 OUTPUT,'A NEW ALGORITHM ? YES=0, NO=1,...,9.'
21 - 21.000 C READ(105,*),NEW
22 - 22.000 C IF(NEW.NE.0)GO TO 1001
23 - 23.000 C OUTPUT,'HOW MANY LEVELS ?'
24 - 24.000 C READ(105,*),NL
25 - 25.000 C OUTPUT,'ENTER No OF LINES'
26 - 26.000 C READ(105,*),LI
27 - 27.000 C OUTPUT,'ENTER ALGORITHM's MATRIX'
28 - 28.000 C DO 23 I=1,LI
29 - 29.000 C 23 READ(105,24)(ALG(I,J),J=1,NL)
30 - 30.000 C 24 FORMAT(100I2)
31 - 31.000 C X=ALG(1,1)
32 - 32.000 1001 OUTPUT,'ENTER No OF EXECUTIONS'
33 - 33.000 C READ(105,*),P
34 - 34.000 C OUTPUT,'ENTER No OF MULTIPLIERS'
35 - 35.000 C READ(105,*),N
36 - 36.000 C OUTPUT,'LIMIT ON AVAILABLE ADDERS? YES=0, NO=1,...,9.'
37 - 37.000 C READ(105,*),JMP
38 - 38.000 C IF(JMP.NE.0)GO TO 696
39 - 39.000 C OUTPUT,'ENTER No OF ADDERS'
40 - 40.000 C READ(105,*),NAD
41 - 41.000 C START 1ST LEVEL OPERATIONS (MULTIPLICATIONS)
42 - 42.000 696 I=1
43 - 43.000 1 IF (I.GT.P) GO TO 11
44 - 44.000 C K=X
45 - 45.000 C J=0
46 - 46.000 2 IF (K.EQ.0) GO TO 12
47 - 47.000 C J=J+1
48 - 48.000 C IF (I.NE.1) THEN
49 - 49.000 C S=0
50 - 50.000 C DO 122 IX=1,I-1
51 - 51.000 122 S=MUL(I-IX,J)+S
52 - 52.000 C IF (S.LT.N) THEN
53 - 53.000 C NI=N-S
54 - 54.000 C ELSE
55 - 55.000 C NI=0
56 - 56.000 C END IF
57 - 57.000 C ELSE
58 - 58.000 C NI=N

```

```

59 - 59.000      END IF
60 - 60.000      IF(K.LE.NI) THEN
61 - 61.000      MUL(I,J)=K
62 - 62.000      ELSE
63 - 63.000      MUL(I,J)=NI
64 - 64.000      END IF
65 - 65.000      IS=0
66 - 66.000      DO 31 IJ=1,J
67 - 67.000      31 IS=MUL(I,IJ)+IS
68 - 68.000      K=X-IS
69 - 69.000      GO TO 2
70 - 70.000      12 DO 3 L=J+1,350
71 - 71.000      3  MUL(I,L)=0
72 - 72.000      I=I+1
73 - 73.000      GO TO 1
74 - 74.000      11 T=J
75 - 75.000      C 1ST LEVEL OPERATIONS COMPLETED
76 - 76.000      C CLEAR ALL ARRAYS
77 - 77.000      DO 21 I=1,65
78 - 78.000      DO 210 J=1,350
79 - 79.000      AD(I,J)=0
80 - 80.000      210 CONTINUE
81 - 81.000      21 CONTINUE
82 - 82.000      C START OPERATIONS OF NEXT LEVEL
83 - 83.000      I=1
84 - 84.000      LV=1
85 - 85.000      REPEAT 313, WHILE(I.LE.P)
86 - 86.000      C CLEAR AUXILIARY MATRIX
87 - 87.000      DO 22 MI=1,10
88 - 88.000      DO 220 J=1,400
89 - 89.000      220 AUX(MI,J)=0
90 - 90.000      22 CONTINUE
91 - 91.000      J=2
92 - 92.000      125 MS1=0
93 - 93.000      DO 25 KR=1,J-1
94 - 94.000      25 MS1=MS1+MUL(I,KR)
95 - 95.000      IF(MS1.EQ.0) THEN
96 - 96.000      J=J+1
97 - 97.000      GO TO 125
98 - 98.000      END IF
99 - 99.000      L=LI
100 - 100.000     REPEAT 28, WHILE(MS1.NF.ALG(L,LV))
101 - 101.000     28 L=L-1
102 - 102.000     AD(I,J)=ALG(L,LV+1)
103 - 103.000     MS3=0
104 - 104.000     DO 29 JX=1,J-1
105 - 105.000     29 MS3=MS3+AD(I,JX)
106 - 106.000     AD(I,J)=AD(I,J)-MS3
107 - 107.000     IF(MS1.EQ.ALG(1,LV)) GO TO 251
108 - 108.000     J=J+1
109 - 109.000     GO TO 125
110 - 110.000     C OPERATIONS OF CURRENT LEVEL COMPLETED, NEXT LEVEL
111 - 111.000     251 NLV=1
112 - 112.000     DO 30 NK=1,400
113 - 113.000     30 AUX(NLV,NK)=AD(I,NK)
114 - 114.000     IF(JMP.NE.0)GO TO 566
115 - 115.000     CALL LIHADD(AD,AUX,ALG,I,NLV,LV,NAD,NJJ)
116 - 116.000     J=NJJ
117 - 117.000     566 REPEAT 321, WHILE(LV.LT.NL-1)
118 - 118.000     J=2

```

```

119 - 119.000      LV=LV+1
120 - 120.000      133 MS4=0
121 - 121.000      DO 33 JP=1,J-1
122 - 122.000      33 MS4=MS4+AUX(NLV,JP)
123 - 123.000      L=LI
124 - 124.000      REPEAT 35, WHILE(MS4.NE.ALG(L,LV))
125 - 125.000      35 L=L-1
126 - 126.000      MS6=0
127 - 127.000      IF(J-1.LE.T) THEN
128 - 128.000      LIM=J-1
129 - 129.000      ELSE
130 - 130.000      LIM=T
131 - 131.000      END IF
132 - 132.000      DO 36 JF=1,LIM
133 - 133.000      36 MS6=MS6+MUL(I,JF)
134 - 134.000      ISE=0
135 - 135.000      136 IF(MS6.GE.ALG(L,1)) THEN
136 - 136.000      IF(ALG(L,LV+1).EQ.0) THEN
137 - 137.000      IF((L.GT.1).AND.(ISE.NE.1)) THEN
138 - 138.000      IF(ALG(L-1,LV).EQ.ALG(L,LV)) THEN
139 - 139.000      L=L-1
140 - 140.000      GO TO 136
141 - 141.000      END IF
142 - 142.000      ELSE
143 - 143.000      ISE=0
144 - 144.000      AUX(NLV+1,J)=0
145 - 145.000      END IF
146 - 146.000      ELSE
147 - 147.000      AUX(NLV+1,J)=ALG(L,LV+1)
148 - 148.000      END IF
149 - 149.000      MS7=0
150 - 150.000      DO 37 JJ=1,J-1
151 - 151.000      37 MS7=MS7+AUX(NLV+1,JJ)
152 - 152.000      AUX(NLV+1,J)=AUX(NLV+1,J)-MS7
153 - 153.000      ELSE
154 - 154.000      IF(L.LT.LI) THEN
155 - 155.000      L=L+1
156 - 156.000      ISE=1
157 - 157.000      GO TO 136
158 - 158.000      ELSE
159 - 159.000      IF(L.EQ.LI) THEN
160 - 160.000      AUX(NLV+1,J)=ALG(L,LV+1)
161 - 161.000      GO TO 370
162 - 162.000      ELSE
163 - 163.000      OUTPUT,'ERROR..I',I,L,MS6,ALG(L,1),J
164 - 164.000      GO TO 99
165 - 165.000      END IF
166 - 166.000      END IF
167 - 167.000      END IF
168 - 168.000      IF(MS4.EQ.ALG(1,LV)) GO TO 331
169 - 169.000      370 J=J+1
170 - 170.000      GO TO 133
171 - 171.000      331 NLV=NLV+1
172 - 172.000      IF(JMP.NE.0) GO TO 321
173 - 173.000      CALL LIMADD(AD,AUX,ALG,I,NLV,LV,NAD,NJJ)
174 - 174.000      J=NJJ
175 - 175.000      321 CONTINUE
176 - 176.000      DO 39 KF=1,J
177 - 177.000      MS8=0
178 - 178.000      DO 38 IE=1,NL-1

```

```

179 - 179.000 38 MS8=MS8+AUX(IE,KF)
180 - 180.000 39 AD(I,KF)=MS8
181 - 181.000 999 IF(AD(I,J).GT.ALG(1,NL)) THEN
182 - 182.000     AD(I,J+1)=AD(I,J)-ALG(1,NL)
183 - 183.000     AD(I,J)=ALG(1,NL)
184 - 184.000     J=J+1
185 - 185.000     GO TO 999
186 - 186.000     END IF
187 - 187.000     JT=J
188 - 188.000     LV=1
189 - 189.000 C OPERATIONS OF CURRENT POINT COMPLETED, NEXT POINT.
190 - 190.000 313 I=I+1
191 - 191.000     OUTPUT,'ALLOCATION PRINTOUT? YES=0, NO=1,...,9.'
192 - 192.000     READ(105,*),JUM
193 - 193.000     IF(JUM.NE.0)GO TO 393
194 - 194.000     DO 40 I=1,P
195 - 195.000     40 WRITE(6,401)(AD(I,J),J=1,JT)
196 - 196.000     401 FORMAT(2X,100I2)
197 - 197.000     WRITE(6,908)
198 - 198.000     908 FORMAT(1X,'ADDTN:',/,1X,'MLTPLs')
199 - 199.000     99 DO 19 I=1,P
200 - 200.000     19 WRITE(6,101)(MUL(I,J),J=1,T)
201 - 201.000     101 FORMAT(2X,100I2)
202 - 202.000     393 POINTS=P
203 - 203.000     RT=POINTS/JT
204 - 204.000     WRITE(6,102),POINTS,JT,RT
205 - 205.000     102 FORMAT(1X,13,2X,'EXECUTIONS IN',2X,I3,2X,'CYCLES',/,2X,F4.2,
206 - 206.000     *2X,'EXECUTIONS/CYCLE')
207 - 207.000     IF(JMP.EQ.0)GO TO 866
208 - 208.000     CALL AMAX(AD,P,JT,MAX)
209 - 209.000     WRITE(6,103),MAX
210 - 210.000     103 FORMAT(2X,'AT LEAST',1X,I2,1X,'ADDERS REQUIRED')
211 - 211.000     NAD=MAX
212 - 212.000     866 CP=(N+NAD)/RT
213 - 213.000     WRITE(6,104),CP
214 - 214.000     104 FORMAT(2X,'COST/PERFORMANCE RATIO :',1X,F5.2,/)
215 - 215.000     OUTPUT,'CONTINUE ? NO=0, YES=1,...,9.'
216 - 216.000     READ(105,*),R
217 - 217.000     IF(R.NE.0) GO TO 1000
218 - 218.000     END
219 - 219.000 C
220 - 220.000 C
221 - 221.000 C THIS SUBPROGRAM COMPUTES THE MAX NO OF REQUIRED ADDERS
222 - 222.000     SURROUTINE AMAX(AD,KP,JT,MAX)
223 - 223.000     INTEGER AD(65,350)
224 - 224.000     N=JT
225 - 225.000     K=KP
226 - 226.000     MAX=0
227 - 227.000     DO 2 J=2,N
228 - 228.000     MAXI=0
229 - 229.000     DO 1 I=1,K
230 - 230.000     1 MAXI=MAXI+AD(I,J)
231 - 231.000     IF(MAXI.GT.MAX) MAX=MAXI
232 - 232.000     2 CONTINUE
233 - 233.000     RETURN
234 - 234.000     END
235 - 235.000 C
236 - 236.000 C
237 - 237.000 C THIS SUBPROGRAM MODIFIES THE ALLOCATION OF THE
238 - 238.000 C ADDITIONS, IN ORDER TO MEET THE REQUIREMENT

```

```

239 - 239.000 C OF LIMITED No OF AVAILABLE ADDERS.
240 - 240.000 SUBROUTINE LIMADD(AD,AUX,ALG,II,NLV,LU,NAD,NJJ)
241 - 241.000 INTEGER AD(65,350),AUX(10,400),ALG(20,10)
242 - 242.000 NA=NAD
243 - 243.000 INLV=NLV
244 - 244.000 L=LV
245 - 245.000 KN=ALG(1,L+1)
246 - 246.000 J=1
247 - 247.000 I=II
248 - 248.000 K=KN
249 - 249.000 REPEAT 10, WHILE(K.NE.0)
250 - 250.000 J=J+1
251 - 251.000 S1=0
252 - 252.000 IF(I.EQ.1) GO TO 7
253 - 253.000 DO 1 IX=1,I-1
254 - 254.000 1 S1=AD(IX,J)+S1
255 - 255.000 7 S2=0
256 - 256.000 IF(INLV.EQ.1)GO TO 77
257 - 257.000 DO 2 IZ=1,INLV-1
258 - 258.000 2 S2=AUX(IZ,J)+S2
259 - 259.000 77 S=S1+S2
260 - 260.000 IF(S.LT.NA) THEN
261 - 261.000 IF(AUX(INLV,J).GT.NA-S) THEN
262 - 262.000 AUX(INLV,J+1)=AUX(INLV,J)+AUX(INLV,J+1)-NA+S
263 - 263.000 AUX(INLV,J)=NA-S
264 - 264.000 END IF
265 - 265.000 K=K-AUX(INLV,J)
266 - 266.000 ELSE
267 - 267.000 AUX(INLV,J+1)=AUX(INLV,J+1)+AUX(INLV,J)
268 - 268.000 AUX(INLV,J)=0
269 - 269.000 END IF
270 - 270.000 10 CONTINUE
271 - 271.000 NJJ=J
272 - 272.000 RETURN
273 - 273.000 END

```

Chapter 4

Proof of Case 1 ($t_i=0$)

Let t_0 be the time an operator uses the input port of a chip to load data produced (fetched) in time t_p , then in the time interval (see Figure 4.9)

$$t_p+t_0, \quad t_p+t_0+t$$

where

$$t = t_x+t_p$$

there will be space for n input operations such that

$$t = nt_0$$

or

$$n = (t_x+t_p)/t_0$$

thus, the number m of operators that can use the port without conflict is

$$m = 1 + (t_x+t_p)/t_0$$

with a maximum arrival rate of $1/t_0$ arrivals per second. The maximum execution rate occurs when

$$t_x+t_p = kt_0$$

Figure 4.9 : Timing Diagram for the Case of $t_i = 0$ (Case 1)

where k is a positive integer.

Proof of Case 2 ($t_0=0$)

Let t_i be the time an operator uses the output port of a chip to input data that is to be processed by the operator in time t_p . Obviously, the maximum number of such operators that can use the port without conflict is

$$m = \left\lceil (t_i + t_p + t_x) / t_i \right\rceil$$

or

$$m = 1 + \left\lfloor (t_p + t_x) / t_i \right\rfloor$$

In this case the maximum rate of $1/t_i$ inputs per second occurs when

$$t_p + t_x = kt_i$$

where k is a positive integer.

Proof of $m_3=0$:

First we prove the following Corollary 2 of Theorem 2.

Corollary 2: $m_3 < m_1$

Proof:

$$m_3 = \left\lfloor \frac{t_{pr} - (n+1)f - t_0}{t_i} \right\rfloor u(k) \quad , \quad k = \left\lfloor \frac{t_{pr} - (n+1)f - t_0}{t_i} \right\rfloor$$

it is

$$\left\lfloor \frac{t_{pr} - (n+1)f - t_0}{t_i} \right\rfloor < \frac{t_{pr} - (n+1)f - t_0}{t_i} < \frac{t_{pr} - n'f - t_0}{t_i}$$

where $n = \lfloor n' \rfloor < n+1$, $(\lfloor n' \rfloor \leq n' < \lfloor n' \rfloor + 1)$

thus

$$m_3 < \frac{t_{pr} - n'f - t_0}{t_i} = m_1$$

or $m_3 < m_1$

Q.E.D.

Now in the case where $t_x = 0$, $m_1 = 1$ and since $m_3 < m_1$ it should be $m_3 = 0$.

For this case the upper bound of R is calculated as follows:

$$R = \frac{1 + \left\lfloor \frac{t_p}{t_o + t_i} \right\rfloor}{t_o + t_i + t_p}$$

since

$$\left\lfloor \frac{t_p}{t_o + t_i} \right\rfloor \leq \frac{t_p}{t_o + t_i} \quad \text{then}$$

$$R \leq \frac{1 + \frac{t_p}{t_o + t_i}}{t_o + t_p + t_i} = \frac{1}{t_o + t_i}$$

thus

$$R \leq \frac{1}{t_o + t_i} \quad (\text{ops})$$

which occurs when $t_p = k(t_o + t_i)$ where k is a positive integer.

A strange situation that could happen here is that it could be required to increase the execution time of the PEs (i.e., delay the execution) in order to achieve the maximum execution rate.

Derivation of Equations (3.6) and (3.7)

By using the following relation

$$\lceil a/b \rceil \leq a/b + 1 - 1/b, \forall a, b \geq 0$$

equation (3.5) becomes

$$\begin{aligned} m_p &\leq (t_p/t_{tr} + 1 - 1/t_{tr})(1 + t_{dt}/t_d) \rightarrow \\ m_p &\leq (t_p/t_{tr} + 1 - 1/t_{tr})(t_{tr}/t_d) \rightarrow \\ m_p &\leq (t_p + t_{tr} - 1)/t_d \end{aligned} \quad (1)$$

The utilization of the m_p PEs is

$$U = t_p m_d / (t_{tr} m_p)$$

or by using equation (3.4)

$$U = (t_p / (t_{tr} m_p)) (1 + \lceil t_{td}/t_d \rceil) \quad (2)$$

The lower bound of U is easily obtained by using equations (3.1) and (2).

Derivation of equations (3.10) and (3.11) follows a similar as above procedure.

END

2 5 0 2 8 6

FIN