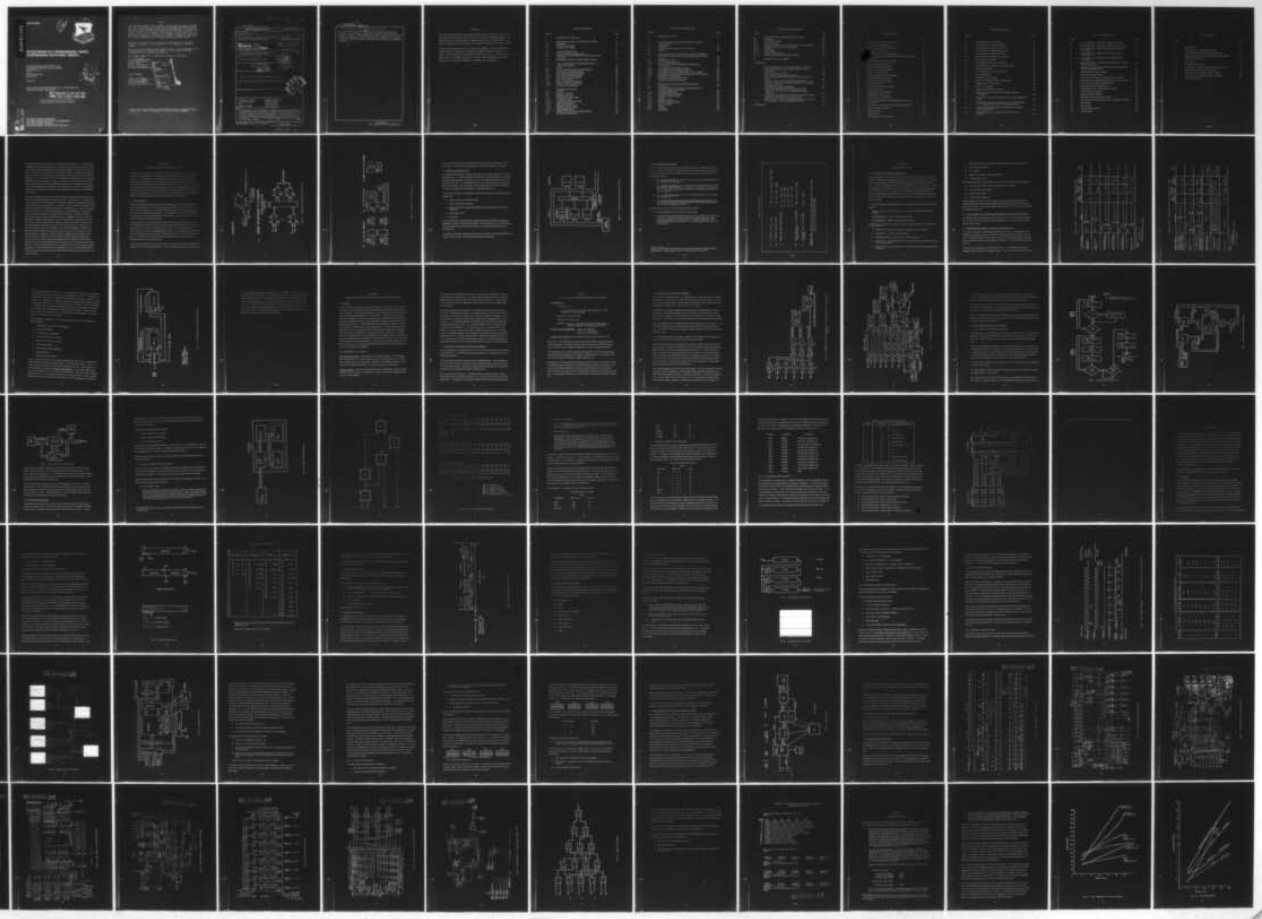
AD-A042 256 RCA GOVERNMENT COMMUNICATIONS SYSTEMS SOMERVILLE N J --ETC F/G 9/2
LSI ELECTRONICALLY PROGRAMMABLE ARRAYS: CONFIGURABLE POLYNOMIAL--ETC(U)
JUN 77 D HAMPEL, K PROST, R L BARRON F33615-73-C-1089
UNCLASSIFIED AFAL-TR-76-228 NL

1 OF 4
AD A042256



AFAL-TR-76-228

LSI ELECTRONICALLY PROGRAMMABLE ARRAYS (CONFIGURABLE POLYNOMIAL ARRAYS)

RCA/ADVANCED COMMUNICATIONS LAB.
GOVERNMENT COMMUNICATIONS SYSTEMS
SOMERVILLE, NJ
AND
ADAPTRONICS, INC.
MCLEAN, VA.

JUNE 1977

FINAL REPORT FOR PERIOD MARCH 1973 thru SEPTEMBER 1976
TECHNICAL REPORT AFAL-TR-76-228

Approved for public release; distribution unlimited

AIR FORCE AVIONICS LABORATORY
AIR FORCE WRIGHT AERONAUTICAL LABORATORIES
AIR FORCE SYSTEMS COMMAND
WRIGHT-PATTERSON AIR FORCE BASE, OHIO 45433

NOTICE

When Government drawings, specifications, or other data are used for any purpose other than in connection with a definitely related Government procurement operation, the United States Government thereby incurs no responsibility nor any obligation whatsoever; and the fact that the government may have formulated, furnished, or in any way supplied the said drawings, specifications, or other data, is not to be regarded by implication or otherwise as in any manner licensing the holder or any other person or corporation, or conveying any rights or permission to manufacture, use, or sell any patented invention that may in any way be related thereto.

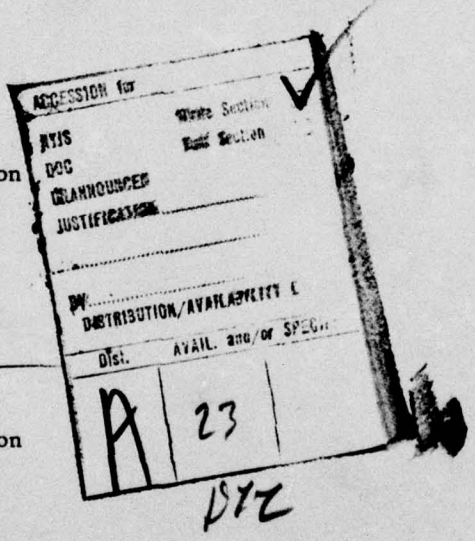
Publication of this report does not constitute Air Force approval of the report's findings or conclusions. It is published only for the exchange and simulation of ideas.

This report has been reviewed by the Information Office (IO) and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nations.

This technical report has been reviewed and is approved for publication.

Cecil W. Gwinn

CECIL W. GWINN
Processor Technology Group
Microelectronics Branch
Electronic Technology Division



FOR THE COMMANDER

Stanley E. Wagner

STANLEY E. WAGNER, Chief
Microelectronics Branch
Electronic Technology Division

Copies of this report should not be returned unless return is required by security considerations, contractual obligations, or notice on a specific document.

AIR FORCE - 8 JUL 77 - 125

(9) Final rept. Mar 73 – Sep 76,

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE

READ INSTRUCTIONS BEFORE COMPLETING FORM

(18) 1. REPORT NUMBER: AFAL-TR-76-228 (19)

2. GOVT ACCESSION NO.

3. RECIPIENT'S CATALOG NUMBER

(6) 4. TITLE (and Subtitle): LSI ELECTRONICALLY PROGRAMMABLE ARRAYS: CONFIGURABLE POLYNOMIAL ARRAYS,

5. TYPE OF REPORT & PERIOD COVERED: Final Draft, March 1973 thru Sept 1976

6. PERFORMING ORG. REPORT NUMBER

(10) 7. AUTHOR(s): ~~RCA~~: D. Hampel, K. Prost, ~~Adaptronics, Inc.~~: R. L. Barron, K. Augustyn, D. Cleveland, ~~M. Whalen~~

(15) 8. CONTRACT OR GRANT NUMBER(s): F33615-73-C-1089 new

9. PERFORMING ORGANIZATION NAME AND ADDRESS: RCA; Advanced Communications Laboratory; Govt. Communications Systems; Sommerville, NJ

10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS: 62204F; 60960108

11. CONTROLLING OFFICE NAME AND ADDRESS: AFAL/DHE-1; WPAFB, OHIO 45433

(11) 12. REPORT DATE: June 1977

(12) 13. NUMBER OF PAGES: 315 p.

14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)

(16) 6096

(17) 01

15. SECURITY CLASS. (of this report): UNCLASSIFIED

15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Distributed Processing	Adaptive Learning Nets
Parallel Prccessing	Ploynomial Arrays
LSI	Array Processing
Signal Processing	Search Algorithms

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The basic objective was the realization of parallel array architectures utilizing LSI distributed logic/memory circuit with the capability of being programmed or trained to perform a variety of signal processing functions. A 32 bit floating point CMOS/SOS cell was developed and a circuit configuration of 8 of these cells realized the desired element. The element computes the following general parametric equation of two inputs and one output. (1a)

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

406 819

1). $Z = W_0 + W_1X_1 + W_2X_2 + W_3X_1X_2 + W_4X_1^2 + W_5X_2^2$

given values of the 'W's' and 'X's'. The thruput computation time ranges from 10 sec to 3.5 sec depending on whether all or part of the above expression is desired in computation. Trade-off studies were carried out with respect to alternate realizations. Bit precision vs speed. Bit precision vs. application and complexity vs cost. The element was realized and demonstrated with respect to computational capabilities.

FOREWORD

This final report describes the work performed under Contract F33615-73-C-1089, "LSI Electronically Programmable Arrays," by RCA Government Communication Systems. A portion of this work was performed by Adaptronics, Inc. under subcontract to RCA. The period of performance covered by this report extended from March 1973 to September 1976.

The AF project monitor was Mr. C. W. Gwinn of AFAL/DHE. The RCA principal investigator was Mr. D. Hampel and the Adaptronics principal investigator was Mr. R. L. Barron. Major contributors to the program from RCA were Mr. K. J. Prost, Mr. R. W. Blasco, Mr. K. E. McGuire, and Mr. N. A. Macina. Major contributors from Adaptronics were D. Cleveland, M. Whalen, K. Augustyn, and A. Mucciardi.

TABLE OF CONTENTS

TABLE OF CONTENTS (contd.)

TABLE OF CONTENTS (contd.)

LIST OF FIGURES

LIST OF FIGURES (contd.)

LIST OF FIGURES (contd.)

LIST OF TABLES

SECTION 1

INTRODUCTION AND SCOPE

This report presents the results of the investigations, analysis, custom LSI development, breadboard design and operation, and supporting tasks on contract F33615-73-C-1089 'LSI Electronically Programmable Logic Arrays."*

The main objective of this investigation was basically, "the realization of parallel array architectures utilizing LSI, distributed logic/memory circuits with the capability of being programmed or trained to perform a variety of signal processing functions."

The following expressions are defined before proceeding with the introductory material.

Configurable Polynomial Array (CPA) — A 2-dimensional array or network of elements capable of being reconfigured (or reconnected) in a variety of ways to describe arbitrarily complex input-output relationships.

Element — A building block (of a CPA) capable of implementing a family of multinomial expressions of its inputs with varying coefficients or weights.

The goals of such elements and the array in which they can be imbedded include:

- the ability to perform a comprehensive family of signal processing functions at higher speeds than is possible with conventional serial computers (using the same basic technologies) by virtue of the inherent parallelism and structure of the CPA concept
- the ability to alter the processing function very easily and rapidly
- the ability to substanially simplify overall software requirements in the application of these techniques to signal processing functions compared to conventional computer methods
- the ability to efficiently implement the complex, non-linear, multinomial expressions associated with Adaptive Learing Networks (ALN's) as well as more conventional linear transformations (such as FFT)

*A more descriptive title for the effort, used throughout this report, is "Configurable Polynomial Arrays," (CPA's). While the arrays are programmable and were specifically conceived to embody LSI technology, they are not to be confused with the "ROM-like" programmable chips used to synthesize Boolean expressions.

- the ability to be used with training algorithms in the synthesis of the networks or arrays for a variety of problems associated with pattern recognition, classification, or discrimination, modeling and identification.

The basic tasks which were addressed, in this investigation included:

1) A study of appropriate architectures, using state-of-the-art technology, necessary to efficiently implement the elements for use in CPA's. This major task area required detailed functional designs for estimating the performance of these elements, tradeoff analysis to compare alternate approaches as a function of calculating precisions and identification of LSI for realization of useful elements

2) The design and fabrication of a custom LSI circuit for use in a variety of versatile elements

3) The design, construction and demonstration of a brassboard element using the special custom chips as well as other existing developmental LSI and commercially available parts

4) The analysis of word length or precision requirements for CPA elements and the implication of these results

5) The applications for which advantageous utilization of CPA's can be made in Avionics signal processing.

Prior to elaborating upon these tasks in the body of this report, a brief perspective of how these arrays differ, in nature, to other array technologies is in order. Arrays for processing signals can be catagorized as either analog (Ref. 1, 2, 3, 4, 5), binary (Ref. 6, 7), or numerical (Ref. 8, 9, 10) depending on the nature of the data and the primitive or "Kernel" function which an element of the array possesses. The analog approaches were representative to some of the early work in arrays useful for classification, feature extraction and pattern recognition. Comprised of threshold elements (basically analog multipliers and summers), they possessed the ability for weight and threshold adjustment. Such arrays of elements could be configured to provide for a variety of signal processing functions. An analog element would provide an output of one state if the weighted sum of the inputs exceeded a given threshold or provide an output of an opposite state otherwise. Analog computer techniques are representative of this type of array.

The binary cellular arrays include such regularly structured systems as associative memories and image processing arrays which can, in effect, perform parallel transformations on binary input patterns.

The numerical element based array is, in effect, the subject of this report. Because of its compatability with digital computers and its flexibility the numerical array consists of arithmetic elements operating on numbers in parallel rather than bits or analog levels and is considered to be the most flexible since it can, in general realize the functions of the analog or binary. However, it may not always be as efficient as one of the other schemes in particular applications. For example, one of its major advantages over analog arrays are the precision and dynamic range it can provide. In some cases where one or two layers or elements (particularly linear elements) suffice in providing a transformation or classification, a numerical unit may not be necessary, although it can be "programmed" to do this. Its compatibility with digital computers, and its versatility are probably the main factors for considering this approach now.

This report organization begins with the element structure, functions and applications, Sections II and III, before going into the hardware aspects. The, Section IV, V, and VI present all major hardware and LSI design aspects. Preliminary analysis of architectural approaches (discussed in the Interim Report, Ref. 11), identified a key LSI circuit for development. This circuit and its use in a demonstrable brassboard are elaborated upon. In each case (Sections II and VI) liberal use is made of supporting appendix material (Appendices A through H) for more detailed and comprehensive explanations and descriptions. Appendix A presents a history and survey of the field of polynomial networks. Appendices A, B, and E describe a procedure for synthesizing these networks from data observations concerning the behavior of a system or process being modeled. Appendix C outlines the principles of networks that use cluster distance-computation primitives in multi-class classifiers. Appendices D, E, and F describe the use of the LSI multiplier-summer chip (while Appendix H describes in detail the actual fabrication and performance of the chip) for CPA's and summarize its potential applications. Appendix G presents results of an application of the nonlinear multinomial primitive to a signal-classification problem arising in unattended ground sensors (used in remote surveillance systems and in target activated munitions). Sections VII and VIII give the analysis and supporting simulations involved in the CPA precision requirements. Section IX discusses how digital filtering can be regarded with CPA's with respect to adaptive learing nets - a new and potentially important application area. The final technical effort reported in Section X describes how pre-processing functions usually required with CPA's, can be accommodated.

SECTION II

PROCESSING ELEMENT AND ARRAY STRUCTURE

A basic element is depicted in Fig 1(a); the figure shows its major control and input and output signals. Such elements are, in general, structured in multi-layered nets, as shown in Fig. 1(b). The interconnection control, which is part of each element, or which appears functionally between successive layers of elements, has inputs which can route the output signals of any one element to the desired input of an element in the next layer. Each element and interconnect switch in the array has sufficient memory for storing the function type it is to compute, the necessary weights or coefficients of its function, and the interconnect data. Hence, this programmable array can be considered as a distributed logic-memory processor capable of rapid reconfiguration and calculation of complex transformations.

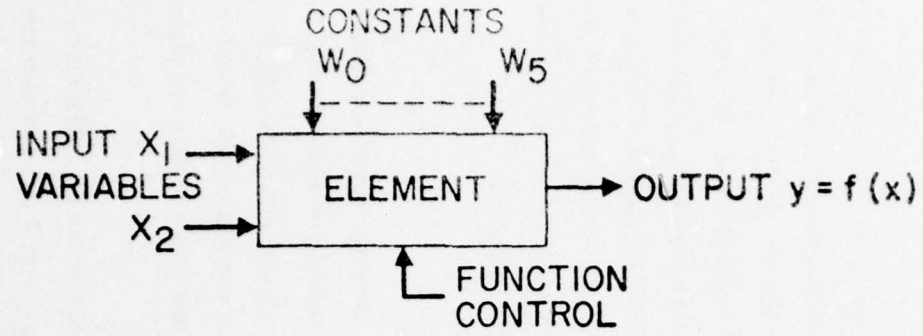
2.1 MULTIPLEXING

Before describing the element requirements and architecture, the three possible configurations of programmable arrays will be explained; these configurations are shown in Fig. 2. In (a), a net of dimension j x k is shown fully populated, with each element containing an m-bit store for necessary function and control. This configuration can process (or transform) j/2 inputs at a time and can be operated in a pipeline mode so that k sets of data are simultaneously operated upon in different layers.

In (b), a single layer of elements can be multiplexed to simulate a whole net. The memory of each element must now have km bits to provide the necessary control as the processor of the element acts, in turn, to realize each of the k layers. For any single problem, the speed is the same as in (a), but pipelining cannot be done. In (c) a single processing element with the jkm bits of storage can be used to process inputs within a layer and then successive layers. Provisions must also be made for storage of intermediate results in (b) and (c); approach (c) will have 1/j the speed of (b).

It is envisioned that arrays of up to 256 elements (or equivalent) could provide the processing capacity for a vast majority of applications. Hence, the design, using either the configuration

(a) ELEMENTS



(b) ARRAY OR NETWORK OF ELEMENTS

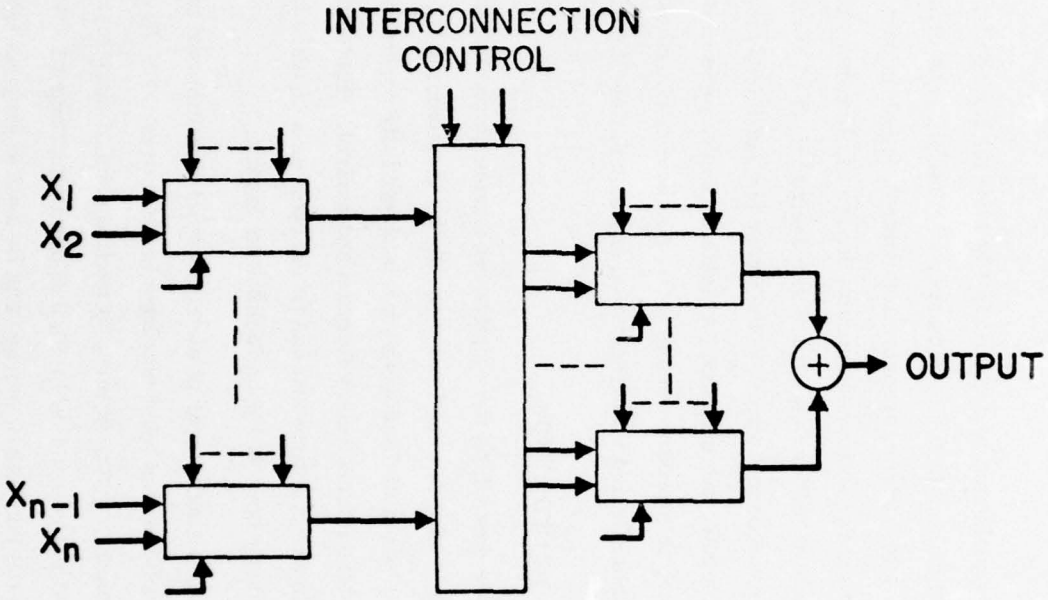


Figure 1. Programmable-Function Array Structure

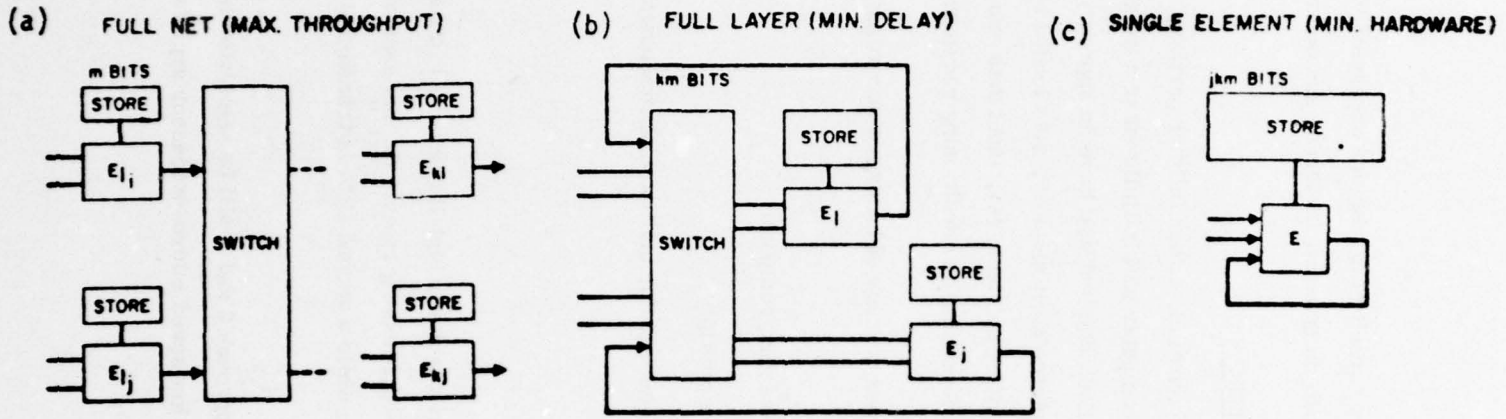


Figure 2. Possible Array Configurations

in Fig. 2(b) or (c), will have the capacity for memory expansion to simulate nets of up to 256 elements. These arrays will, in general, be loaded and controlled by a computer.

2.2 LOADING AND CONTROLLING

The loading and controlling of a parallel, either fully or partially populated array is elaborated upon in Fig. 3. The host computer will initially enter a weight vector(s) via a common bus and element addressing. (Each element will have an appropriate decoder as part of its interface control logic.) The interconnect memory, also loaded by the host, controls the inter-layer switches. After "setting up" an array, input data can be derived directly from its source or through the host, depending on conditioning required, etc.

A CPA, then, can embody various degrees of hardware according to the following hierarchy of approaches:

1) use of all software in existing computer

2) software with hardware multiply

3) a dedicated hardware element (for multinomial implementation) used in a multiplexed mode

4) multi-element array

5) multi-arrays

Increasing hardware complexity results in high thruputs and introduces the possibility of pipeling for yet higher effective processing rates. (In this case one CPA can be performing pre-processing on a set of data while a second CPA is transforming the previous pre-processed data.)

The relative performance of approach 2 and 3 will be seen later in the tradeoff analysis with respect to both high speed and low speed micro-processor application.

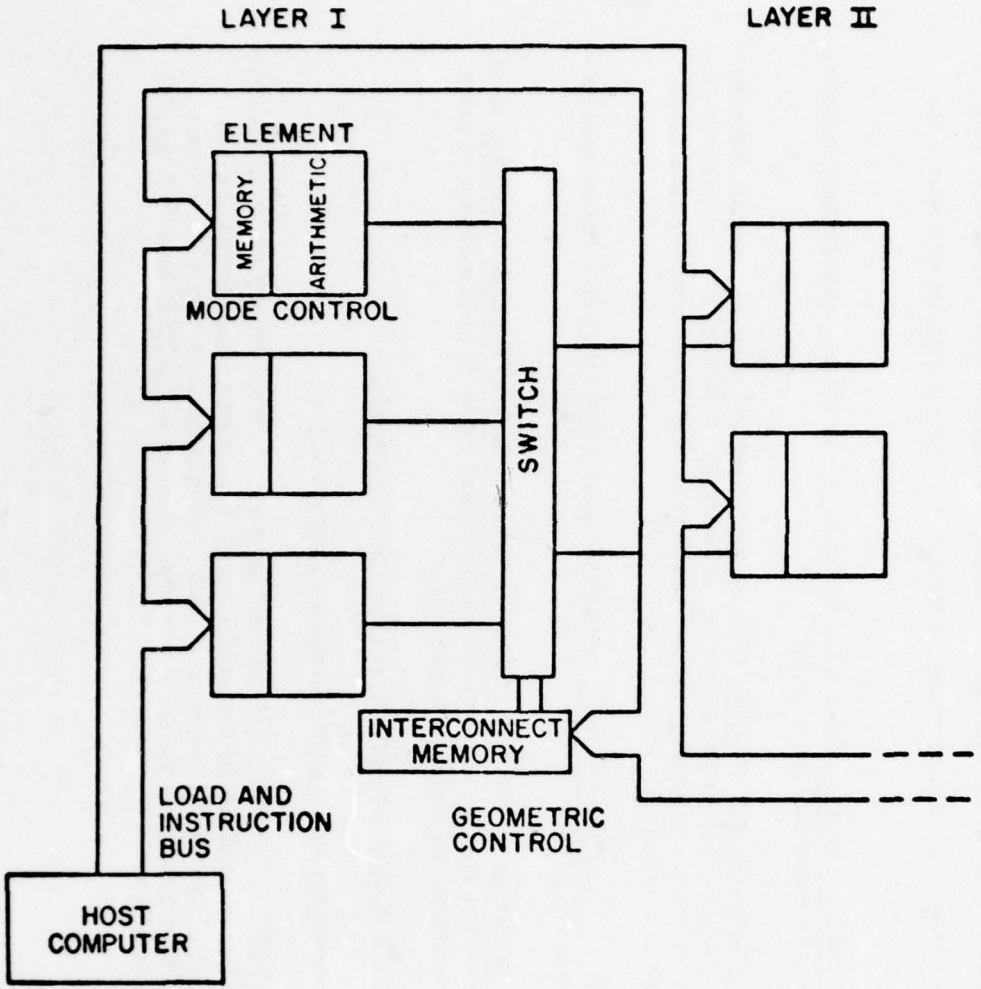


Figure 3. Loading and Controlling a Parallel Array

2.3 FUNCTIONS OF AN ELEMENT

Table 1 lists the five basic expressions realizable by the element. Although any one of a number of kernels or primitives (or families thereof)* could be chosen, in theory, to satisfy the requirements for an element useful in CPA's, the basic list shown was chosen for the following reasons:

1) P1: Nonlinear Multinomials, valuable for nonlinear filters, discriminant functions, estimation networks, and models.

2) P2: Multilinear Multinomials, used in multilinear and nonlinear filters, discriminant functions, estimation netowrks, and models. Providing P1 automatically provides the same computational power required for the simultaneous calculation of two P2 expressions.

3) P3, P4: Linear Multinomials, used in FFT's transversal filters, linear discriminant functions, etc.

4) P5: Component of Normalized Distance Measurement or Recursive Division, used to determine membership or non-membership of a data word in a cluster of prior measurements belonging to a given class.

Characteristics of the above family of primitive functions include:

- the interchangeability of the coefficients and variables
- a variable number of multiply operations; 3 for P5, 4 for P3 and P4, 8 for two P2's and 8 for P1. (While the terms of P1 could have been grouped to require only 5 multiply operations, the element could then only handle a single P2. Speeds would have been about the same since two levels of multiplication would have still been required).

*While any single function would suffice for an element, choosing from amongst a family usually imparts a higher spped for a given element in a given array position.

TABLE 1. CPA FUNCTIONS

P1:	6 TERM POLY (FOR USE IN ALN)	$Y = W_0 + W_1X_1 + W_2X_2 + W_3X_1X_2 + W_4X_1^2 + W_5X_2^2$
P2:	4 TERM POLY (FOR USE IN ALN)	$Y = W_0 + W_1X_1 + W_2X_2 + W_3X_1X_2$
P3:	COMPLEX BUTTERFLY (FOR USE IN FFT)	$P_R' = P_R + (W_RQ_R - W_IQ_I)$ $P_I' = P_I + (W_RQ_I + W_IQ_R)$ $Q_R' = P_R - (W_RQ_R - W_IQ_I)$ $Q_I' = P_I - (W_RQ_I + W_IQ_R)$
P4:	LINEAR TERMS (FOR USE IN TRANSVERSAL DIGITAL FILTERS)	$Y = W_1X_1 + W_2X_2 + W_3X_3 + W_4X_4$
P5:	2 TERM POLY (FOR USE IN RECURSIVE DIVISION)	$Y = W_0X_2 - X_1(X_2)^2$

EXECUTION TIMES 10MHz clock, 10us for 32 bit floating point (P1, 2 P2's or P5)
5.2us for 24 bit fixed point (P3 or P4)
3.5us for 16 bit fixed point (P3 or P4)

SECTION III

APPLICATIONS OF CPA's

3.1 AVIONICS COMPUTATION REQUIREMENTS

Avionics computation requirements arise in two major areas (Reference 12): (a) general purpose computing tasks, typically characterized by I/O sampling rates below 500 Hz and a variety of logic, control, and arithmetic operations performed at throughput rates below 500 K - Ops; and (b) signal processing tasks, typically having I/O rates in excess of 1 MHz, "a relatively narrow regular flow of arithmetic operations," and high computational speeds in the 10-100 MHz range. As stressed in the reference, on-going developments in LSI arrays for signal processing may produce economic and performance benefits comparable to those demonstrated in general purpose computation by LSI micorprocessor and microcontroller elements.

For airborne signal processing applications, the reference cites requirements in the following areas:

Radar — air-to-air and air-to-ground modes, including synthetic aperture ground mapping.

Electronic Warfare — signal sorting and classification.

Communications — image/waveform coding and decoding.

The algorithms of particular importance in airborne signal processing are, in accordance with the reference:

- Digital Fourier transforms and inverse transforms up to 2,048 points
- Digital filters — recursive and nonrecursive
- Weighting functions — cosine-squared, Taylor, Hamming, etc.
- Correlations — serial and parallel, with various levels and combinations of source and reference signals
- Walsh functions, Hadamard transforms, and related waveform/image coding transformations

- Adaptive predictive coding and other bandwidth compression techniques for data
- Nested polynomial functions
- Look-up tables
- Integration, averaging, and standard deviation
- Coordinate conversion.

The following algorithms are added by the authors to the above signal processing functions:

- Recursive means and covariance matrices of vector waveforms
- Convolutions and deconvolutions
- Data cluster screening
- Complex Boolean arithmetic

It is believed that CPA's are best suited to signal processing applications because of the functions implemented and the extremely high processing speed of distributed-function networks of CPA devices. Accordingly, attention in this report will be directed primarily toward the signal processing area.

3.2 FULFILLMENT OF AVIONICS SIGNAL PROCESSING REQUIREMENTS WITH CPA's

The Avionics signal processing requirements enumerated in 3.1 can all be fulfilled using the P1, P2, P3, P4, and/or P5 primitive functions defined in 2.3. Table 2, Parts 1-3, presents the details of this.

3.3 RELATIONSHIP TO ADAPTIVE LEARNING NETWORKS (ALN's)

As one of the main factors influencing the development of CPA's, the ideas behind their use in Adaptive Learning Nets (ALN's) besides known transformations will be briefly discussed. Fig. 4 is a flow chart showing how an input data set is used to train and evaluate a net in the process of "fitting a surface or determining a transformation" for discrimination, classification, etc.

In this process, the input data is initially sub-divided into two independent main groups - a fitting and selection subset and an evaluation subset. These groups contain some parameters

TABLE 2, PART 1 - ALGORITHMS FOR AVIONICS SIGNAL PROCESSING

NAME OF ALGORITHM	TRANSFORM	KERNEL (ELEMENT) FUNCTION(S)	NUMBER OF EXECUTIONS USING ELEMENT(S)	NUMBER OF LAYERS IN NETWORK(S)	PART 1
FAST FOURIER TRANSFORM (FFT)	$X(n) = \sum_{i=0}^{N-1} x(i)\exp(-2\pi jin/N)$	P3	$\frac{N}{2}\log_2 N$	$\log_2 N$	
INVERSE FOURIER TRANSFORM	$X^*(n) = \frac{1}{N}\sum_{i=0}^{N-1} x(i)\exp(2\pi jin/N)$	P3	$\frac{N}{2}\log_2 N$	$\log_2 N$	
FFT CONVOLUTION OF FINITE-LENGTH WAVEFORMS	$y_{CON}(i) = \frac{1}{N}\sum_{n=0}^{N-1} X_1(n)X_2(n)\exp(2\pi jin/N)$	P4	$4N + 12N\log_2 N$	$1 + 2\log_2 N$	
FFT DECONVOLUTION OF FINITE-LENGTH WAVEFORMS	$y_{DEC}(i) = \frac{1}{N}\sum_{n=0}^{N-1} \frac{X_1(n)}{X_2(n)}\exp(2\pi jin/N)$	P4	$(2N)^* + 4N + 12N\log_2 N$	$(1)^* + 1 + 2\log_2 N$	
FFT CORRELATION OF FINITE-LENGTH WAVEFORMS	$z(i) = \frac{1}{N}\sum_{n=0}^{N-1} X_1(n)X_2^*(n)\exp(2\pi jin/N)$	P4	$4N + 12N\log_2 N$	$1 + 2\log_2 N$	
HADAMARD TRANSFORM	$X(n) = \sum_{i=0}^{N-1} x(i)\mathrm{sgn}[\exp(-2\pi jin/N)]$	P4	$4N\log_2 N$	$\log_2 N$	

* 2N divisions involving 1 layer

LEGEND:

i = time sample

n = frequency sample

capital letters = complex variables

TABLE 2, PART 2 - ALGORITHMS FOR AVIONICS SIGNAL PROCESSING

NAME OF ALGORITHM	TRANSFORM	KERNEL (ELEMENT) FUNCTION(S)	NUMBER OF EXECUTIONS USING ELEMENT(S)	NUMBER OF LAYERS IN NETWORK(S)	PART 2
WEIGHTING FUNCTIONS, INTEGRATION, AVERAGING, TRANSVERSAL FILTERS, AND LINEAR DISCRIMINANT FUNCTIONS	$y(i) = w_0 + \sum_{k=1}^{N} w(k)x(i-k)$	P4	$\sim\frac{N}{3}$	$\log_4 N$	
ADAPTIVE PREDICTIVE CODING	$y(i) = \frac{1}{N}\sum_{k=1}^{N} a(k)y(i-k) + \frac{1}{N}\sum_{k=1}^{N} b(k)x(i-k)$	P4	$\sim\frac{2N}{3}$	$\log_4 N$	
(1) STANDARD DEVIATION (2) RECURSIVE MEANS AND COVARIANCE MATRICES OF VECTOR WAVEFORMS	(1) $\sigma_x^2 = \frac{1}{N-1}\left\{\sum_{i=1}^{N} x^2(i) - \frac{1}{N}\left[\sum_{i=1}^{N} x(i)\right]^2\right\}$	(1) P4 (2) SEE APPENDIX 1	(1) $\sim\frac{2N}{3}$ (2) SEE APPENDIX 1	(1) $1 + \log_4 N$ (2) SEE APPENDIX 1	
MATRIX OPERATIONS AND COORDINATE CONVERSIONS	Example: $\underline{A} = \underline{\underline{M}}\,\underline{B}$	P4	$\sim\frac{N^2}{3}$	$\log_4 N$	$\underline{\underline{M}}$ is an NXN Matrix
(1) NESTED POLYNOMIAL FUNCTIONS (2)LOOK-UP TABLES	Polynomials and multinomial expansions are synthesized (from data) via Adaptive Learning Network (ALN) procedures to obtain (1) accurate nonlinear approximations for univariate and multivariate transformations, and (2) efficient storage, retrieval, and interpolation of tabular data.				P1 and P2 SPECIFIC TO SOLUTION USED
DIGITAL FILTERS	SEE SECTION 10	P4, P1, or P2	SEE SECTION 10	SEE SECTION 10	

LEGEND:
i = time sample
n = frequency sample
capital letters = complex variables

(2) See Appendix I.

TABLE 2, PART 3 - ALGORITHMS FOR AVIONICS SIGNAL PROCESSING

NAME OF ALGORITHM	TRANSFORM	KERNEL (ELEMENT) FUNCTION(S)	NUMBER OF EXECUTIONS USING ELEMENT(S)	NUMBER OF LAYERS IN NETWORK(S)	PART 3
DATA CLUSTER SCREENING (CNET)	$D^2_{km} = \sum_{j=1}^{N} (x_j - \bar{x}_{jkm})^2/\sigma^2_{km}$	P5, P4	y_4: $N \sum_{k=1}^{K} M_k$ y_{1B}*: $\frac{N}{3} \sum_{k=1}^{K} M_k$	$1 + (\log_4 N)^*$	
BOOLEAN ARITHMETIC	All 16 logic functions of two binary variables.	P2	1	1	

LEGEND:

j denotes component of N-dimensional measurement vector,
k denotes class (K classes total),
m denotes cluster (i.e., mode) within a given class (M_k modes per class).

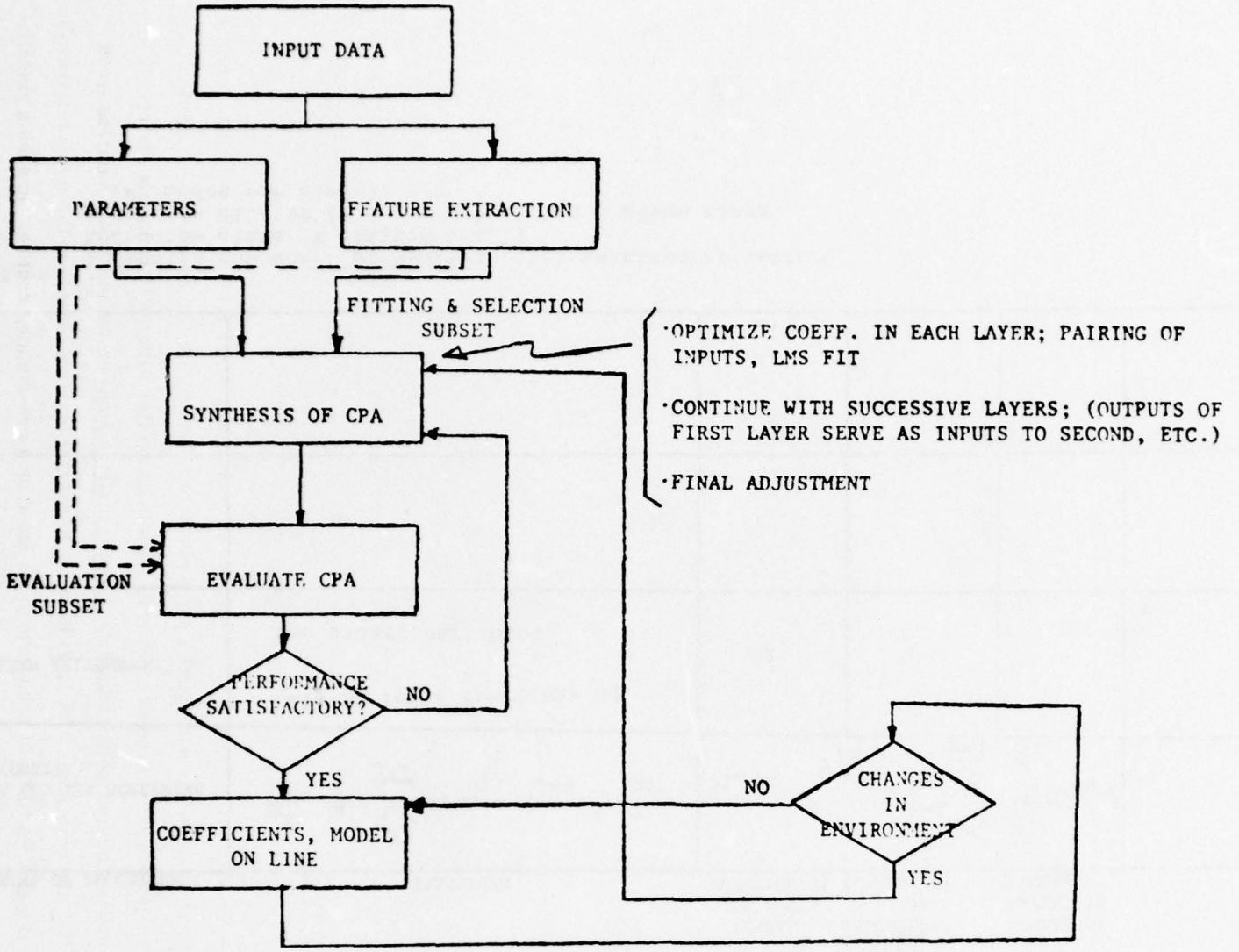


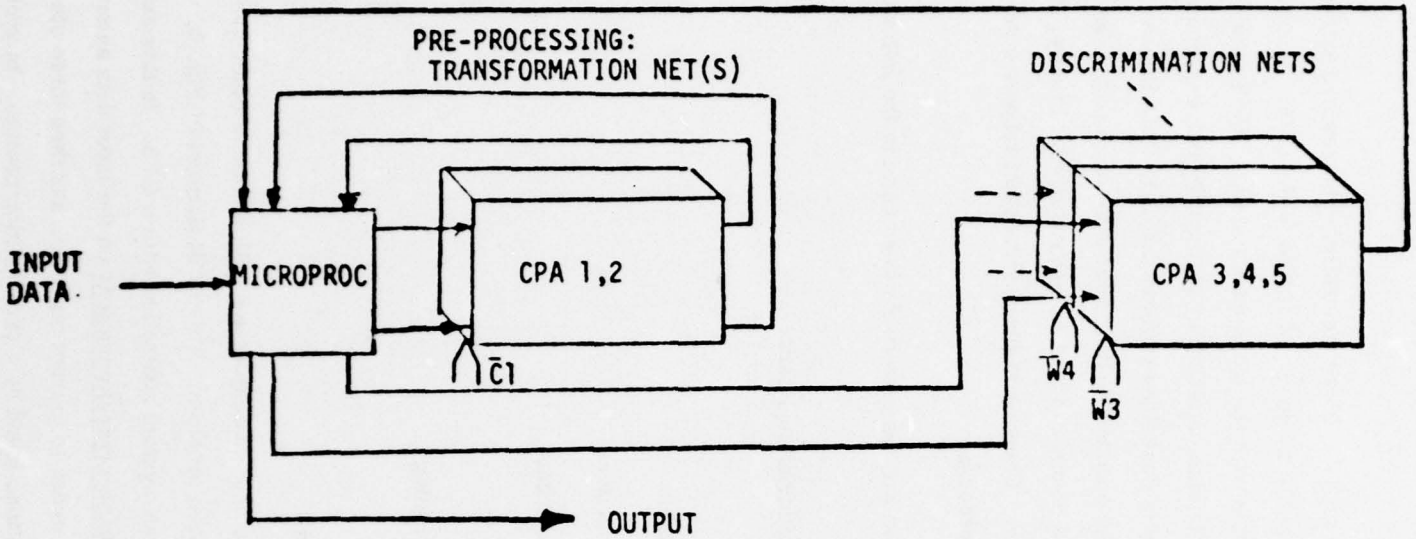
Figure 4. Adaptive Learning Method

which are used directly as inputs to a CPA. More generally, features are extracted from the input data, providing other frequently more useful inputs to the CPA. A combination of these inputs is then used in conjunction with the techniques listed, in the synthesis of a CPA network, i.e., optimization of coefficients, element by element by pairing inputs, followed by multi-layer synthesis, etc. This is described in more detail in Ref. 17, and in Appendix A. Evaluation of the CPA with the independent testing subset is then used to determine if re-adjustment of the coefficients and connectivity is in fact required. Finally, the net is placed "on line" for intended function. Changes in the environment trigger a re-synthesis routine, to maintain a "fit" for the process.

Such methodology, (Appendices A and F), has effectively been used in the following aerospace applications:

1. nondestructive inspection of structural parts
2. trajectory predictions
3. target signature classifications
4. radar refractive index corrections
5. detection of remote nuclear events
6. voice data processing
7. reconnaissance image processing
8. electronic warfare
9. avionics information systems

Once a net has been trained it can be used in conjunction with known or other synthesized net configurations in the solution of a given problem. This is illustrated in Fig. 5, where a micro-processor is used as an overall system controller with a CPA. In this example, the CPA is depicted as performing two known transformations on the input data samples (such as FFT and cepstrum; CPA 1, 2) in order to extract features, and then three other (generally non-linear CPA 3, 4, 5) transformations found by a pre-training routine. In such a classification system, the micro controls the input samples to the CPA's and loads or instructs the



-SAMPLE CONTROL
-FEATURE EXTRACTION
-DECISION LOGIC

Fig. 5. CPA Application to Classification Systems

use of the appropriate coefficient or weight vectors. The output(s) of any CPA may be further processed by the micro, prior to entry into the next CPA. The pre-processing CPA outputs feed the micro for the overall feature extraction phase. (The micro might be performing statistical analysis or other routine data reduction for optimum utilization of the CPA's). Now the CPA is ready to act in its role of a discriminator and can be loaded in turn with the coefficients and interconnectivity pre-stored either in a micro ROM or in the CPA memory itself. The discrimination outputs are again fed to the micro for relatively simple decision logic to form the basic classification.

SECTION IV

BASIC ARCHITECTURAL APPROACHES FOR CPA IMPLEMENTATIONS

4.1 GENERAL

In order to achieve a high degree of applicability of the CPA technology, a fast, simple element with the capability of being stacked, in building block fashion, to realize complex arrays is necessary. A self-contained element on a chip, with the speed and precision necessary for many practical applications is not yet possible. Even recent micro-processor advances, (especially where the micro's architecture was designed for signal processing), although approaching such a goal, require substantial complexity especially when used in the high precision areas. The basic alternatives will be discussed in relation to the technology required to effectively support each architecture. First, each alternative must be viewed with respect to the precision with which the element's multinomial is calculated. Since a wide range of application was considered, precisions ranging from 8 to 48-bits, and, the use of floating point as well as fixed point was considered. Hence, the performance factors for all approaches are estimated as a function of bit precision. There are basically two ways to build an element (aside from a micro-processor based architecture); one centered around the use of a serial-parallel type multiplier(s) and the other centered around the use of all-parallel type multipliers.

The basic distinctions are as follows:

Serial-parallel multiplier - has provision for pre-loading and storing an n bit multiplier under a clock control. The product appears serially, least significant bit first, and the most significant bit after 2n clock cycles. Besides the n bit multiplier register, the chip contains n full-adder and latch stages, as well as some auxiliary control logic.

Parallel multiplier - accepts the n-bit multiplier and n-bit multiplicand words in parallel, and produces the 2n bit product in parallel as a result of asynchronously rippling through an array of n^2 full-adders.

While the all-parallel multiplier is much faster in performing a multiplication, it involves substantially more complex chip(s). The serial-parallel multiplier approaches equivalent throughputs for the polynomial element by using several simpler chips. It also permits the use of serial adders rather than high speed parallel adders, and leads to simplified interconnect problems.

The multiplier technology has progressed very rapidly in the past three to four years for both types of multiplier implementations. The unit developed for use in the CPA (Ref. 18 and included in Appendix H) was followed by a comparative analysis of multipliers (Ref. 19) and developments of small bit capacity, bipolar serial-parallel multipliers (Ref. 20, 21). The latter, now commercially available is considered in the tradeoff analysis of alternate architectural approaches to follow. In the parallel multiplier realm, two large capacity LSI versions were considered: an 8 X 8 SOS (Ref. 22) and a 16 X 16 Bipolar (Ref. 23). Smaller (4 X 4 arrays or less such as in Ref. 24) higher speed units were not considered because of substantially greater overall complexity of the resultant element and because of availability and cost. Such units, characterized by multiplication times of less than 40 ns for 8 X 8 calculations, would ultimately impact the design of very high speed elements, requiring different architectures, and using predominantly ECL parts and bi-polar memories.

4.2 CUSTOM CMOS/SOS SERIAL-PARALLEL MULTIPLIER

The characteristics of the multiplier chosen for use in the mantissa processor of an element is summarized here. A paper with the complete description of its design and operation is included in Appendix H.

The chosen design fits the basic requirements of an efficient element. This multiplier and its use within the element has furthermore shown a generally optimum approach as will be seen in the tradeoff analysis, particularly for high precision calculations (even though alternate circuits and/or architectures excell for certain criteria).

Table 3 summarizes the chip data. It should be pointed out that the top speed - 15 MHz - falls somewhat short of predicted simulation (Ref. 11) values because of the extra delays in the full adder. (This design was in fact partly responsible for the excellent packing density

TABLE 3

CUSTOM CMOS/SOS MULTIPLIER PERFORMANCE SUMMARY

PERFORMANCE:

FUNCTION: ax + b

a, x, b are up to 24 bits in length. Taps provided at 8, 16 bits. Cascadable for greater accuracy.

CHIP SIZE: 155 mil X 170 mil

NUMBER OF DEVICES: 1750

PACKAGE: 16-pin; (3, 8-bit registers for serial/parallel loading of multiplier; multiplicand and addend loaded serially)

TYPICAL POWER DISSIPATION: 5 mw, 5v, 5 MHz clock
80 mw, 10v, 10 MHz clock
300 mw, 15v, 15 MHz clock

ENERGY CONSUMPTION (FOR MULTIPLYING TWO 16-BIT NUMBERS): 64nJ

on the chip.) A second generation circuit would not only result in an improved speed (we would consider this a worthwhile trade with respect to chip area), but would incorporate some extra control logic to minimize external parts count in the mantissa processor.

This multiplier has also resulted in efficient implementations of two other types of signal processing subsystem besides the CPA element. These include two FFT processors (Ref. 25, 26) and a Quadrature Demodulator-Digital Filter (Ref. 27).

Descriptions of the basic architectural alternatives will now be given. One based on the use of the serial-parallel multiplier (word parallel, bit serial processor) and two based on the use of the all parallel multiplier (word serial, bit parallel processors). In the latter case we consider a micro-processor controller within the element as well as the hard-wired case. The comparative performance of all alternatives versus precision is given in Section VI.

4.3 WORD PARALLEL-BIT SERIAL APPROACH

The heart of the polynomial element - the arithmetic unit or mantissa processor - using the custom chips is shown in Fig. 6. With each cell being one of the custom chips, cell 1 functions as a delay register, cells 2 through 9 generate the product terms (the processor shown is configured to implement P1), while cell 10 serves as register.

In this approach, generation of different polynomials involves reconfiguring the cell interconnections. This actually simplifies the control circuitry, as the controller must only provide control signals to the interconnection gates (not shown in Fig. 6) and output a single clock burst. This will be seen in more detail in Section V when the brassboard is described.

The word-parallel processor modified for floating-point calculations is shown in Fig. 7. As in the fixed-point processor, the polynomials are generated by proper interconnection of cells. The function of the cells in the mantissa section is identical to that of the corresponding cells in Fig. 6.

An overflow position detector and register, scalers, a left-justifier, and an exponent processor are provided to perform the necessary floating-point functions.

Since the polynomial terms are added together in parallel, scaling of these terms must be done in parallel. The scaler information in this case represents the difference between the largest of the term exponents and the exponent for the given term. Since this number is always non-negative, the scalers although there are six of them, are relatively simple devices, and consists of pre-settable down counters. During the scaling cycle, the counters are counted down to zero and a clock is enabled to the various cells as long as that cell's scaler is non-zero. When any scaler reaches zero it turns off its clock. The final addition is then performed on all of the individual terms.

After the final addition the mantissa must be left justified. This is accomplished as follows. The entire mantissa is shifted into a serial register and the present incoming bit is compared with the bit received previously. If both bits are the same, a counter is advanced by one, and if they differ, the counter is reset to zero. The mantissa plus 5 additional bits, (allowance for maximum overflow), are shifted into the register and counter circuit. At

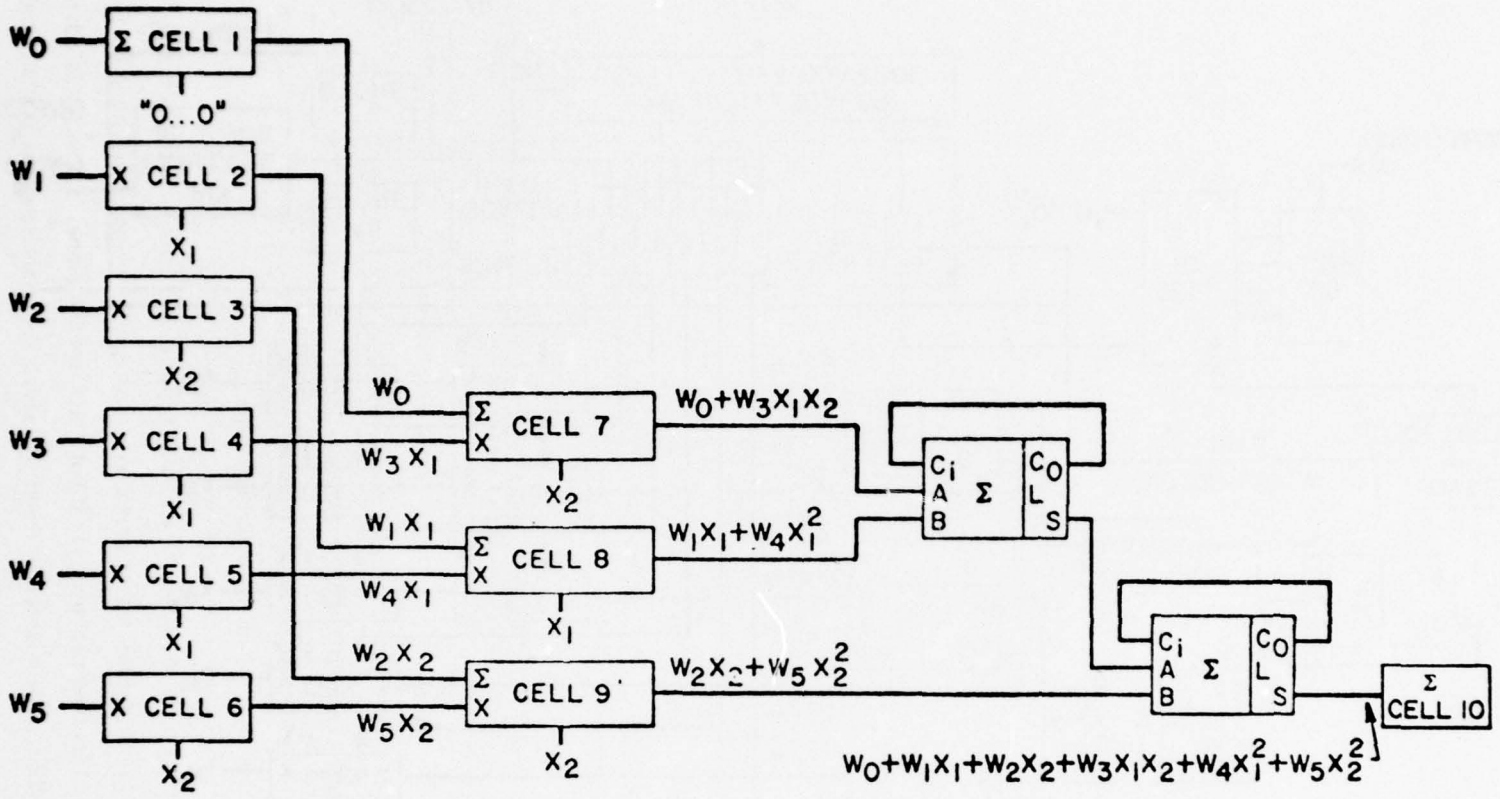


Fig. 6. Arithmetic Unit of Programmable Array using CMOS/SOS 24-Bit (ax+b) Module

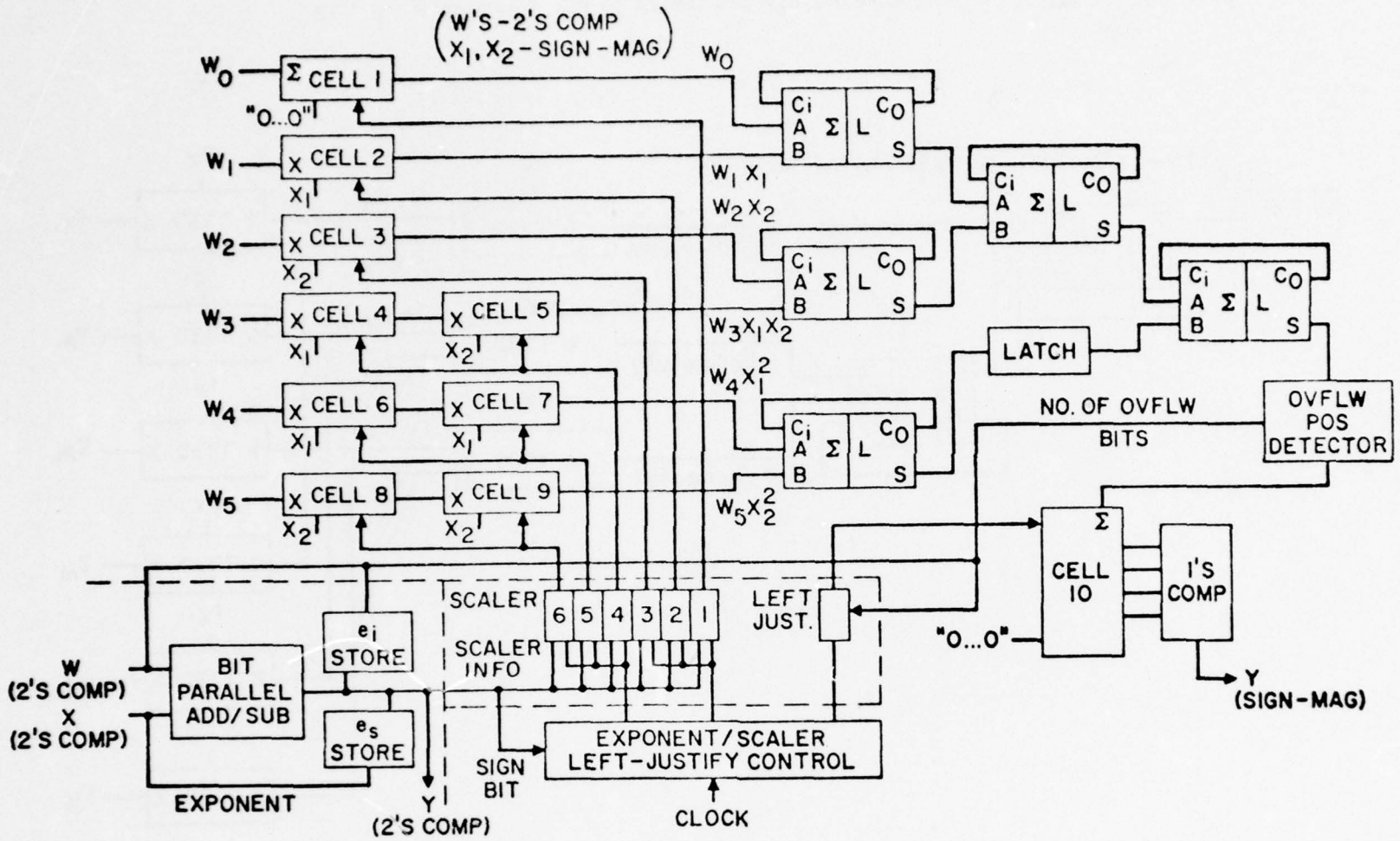


Fig. 7. Word-Parallel Floating-Point Processor

the end of this time the number in the counter indicates the position of the MSB in the mantissa. This number is then transferred to the exponent circuits so that it can calculate the final exponent term and is also used to left or right justify the mantissa.

The exponent section consists of a bit-parallel adder/subtractor (required so that all term exponents are available before the mantissas are added together), two memories, and control logic. The control logic implements the flowchart of Fig. 8.

The use of the ones complementor along with the truncation of the least significant bits will cause a processor error of less than one mantissa LSB for either the bit/word-serial or the word-parallel floating point processors.

4.4 WORD SERIAL-BIT PARALLEL APPROACH

4.4.1 BIT-PARALLEL MULTIPLIER APPROACH

The architecture of a bit-parallel arithmetic processor is shown in Fig. 9. For fixed-point calculations, a bit-parallel multiplier and a bit-parallel accumulator form the heart of the processor. Gating is provided to allow calculations of second- and third-order product terms.

Latches at the multiplier and adder outputs provide synchronous operation of the processor. A typical micro-sequence of operations for a 6 term polynomial would be as shown in Table 4. Individual sequences for the various polynomial types would be stored in program memory and called up when required. Execution time would be basically dependent upon the cycle time of the multiplier and will be considered in the tradeoff analysis.

For floating-point calculations, two parallel scalers, an overflow detector, and an exponent processor must be added.

The scalers align the mantissas of the two floating point numbers to be added so that bits of equal significance are added together.

The overflow detector determines the position of the most-significant bit (MSB) in the output mantissa, so that this mantissa may be left-justified before storage in the data memory.

Function:

1. Find largest exponent term
2. Align all other terms accordingly.

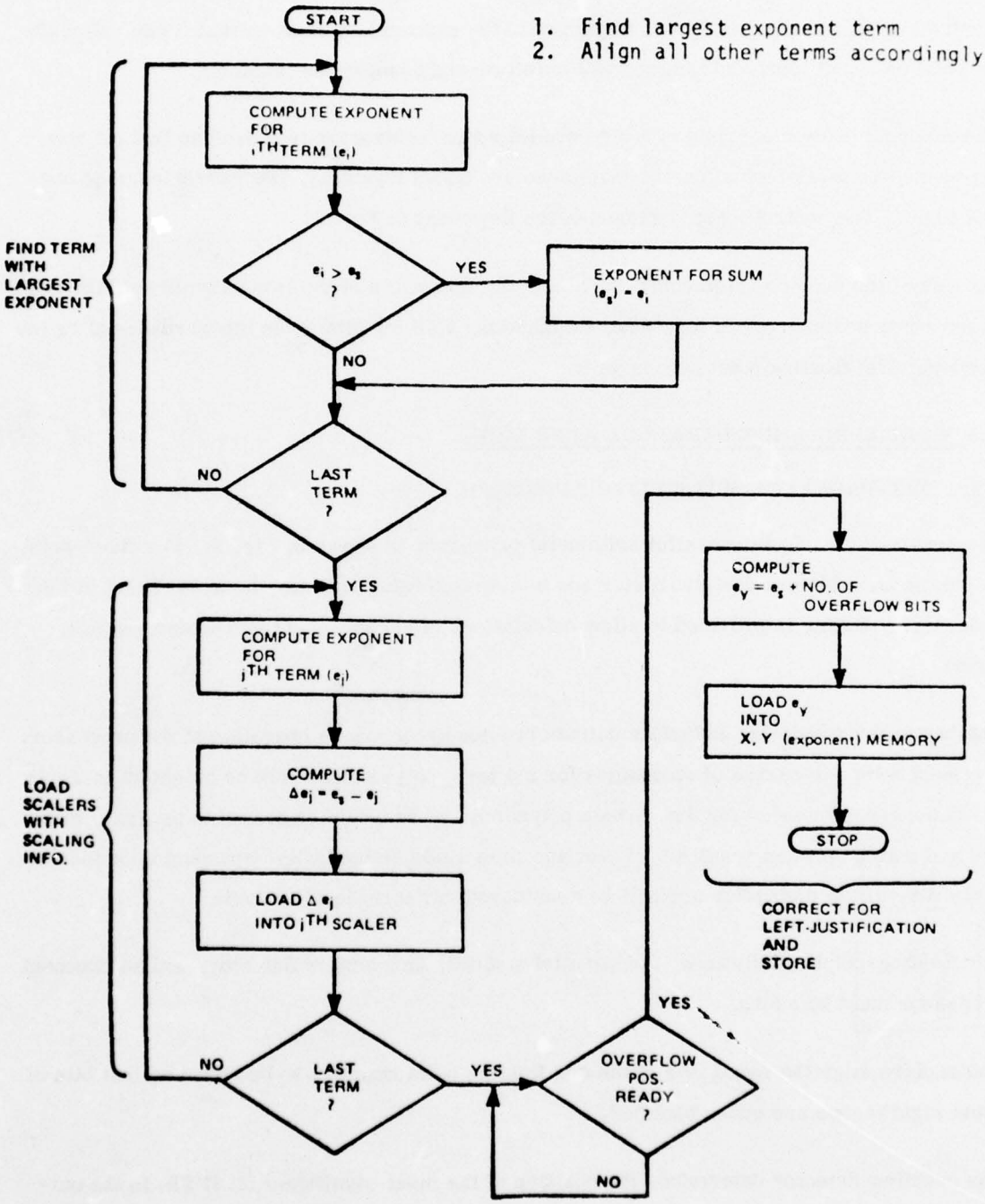


Fig. 8. Exponent Processor Flowchart

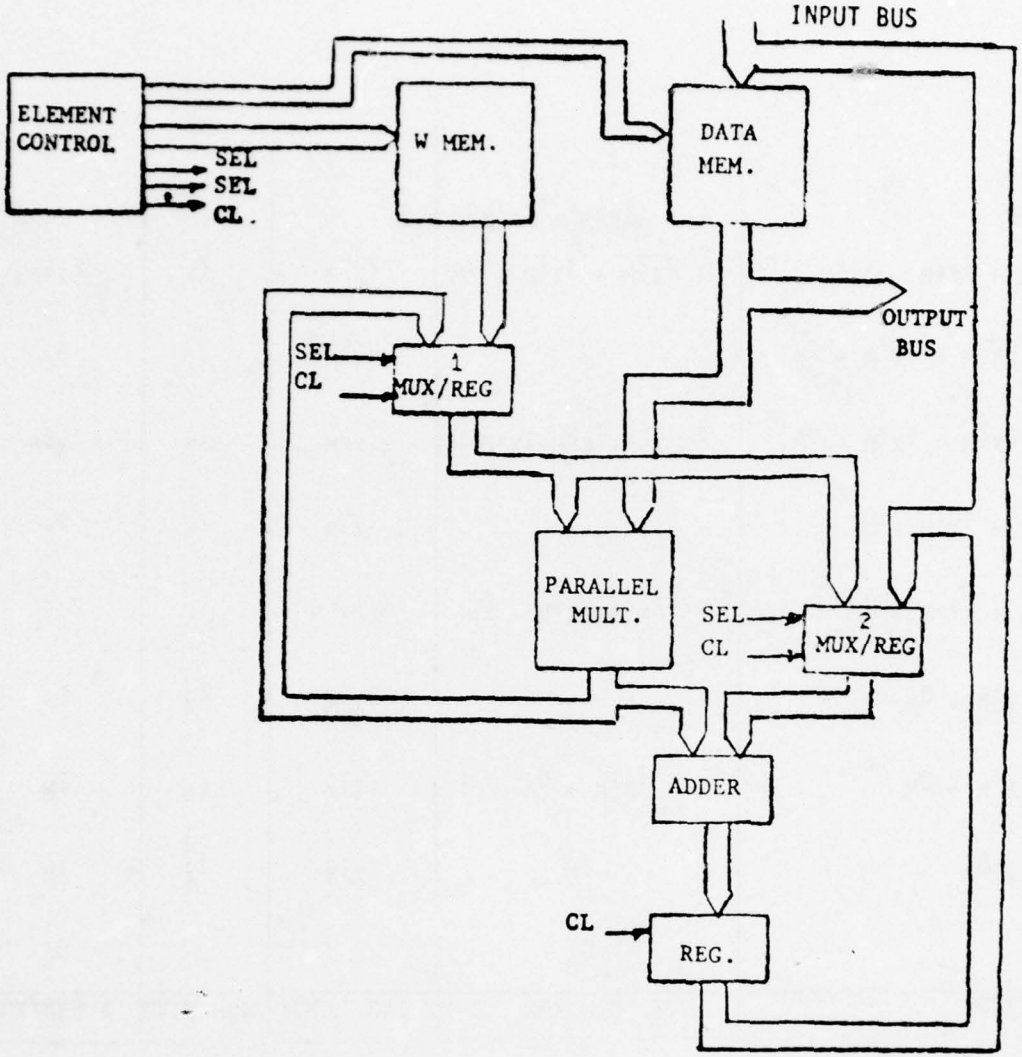


Fig. 9. Bit-Parallel Element Architecture

TABLE 4. SEQUENCE OF OPERATIONS FOR BIT-PARALLEL ELEMENT ARCHITECTURE

STEP	W mem.	MUX/REG 1	DATA MEM.	MULT. OUT	MUX/REG 2	ADDER OUT
1	W_0	W_0	-	-	W_0	-
2	W_1	W_1	X_1	W_1X_1	W_0	$W_0 + W_1X_1$
3	W_2	W_2	X_2	W_2X_2	$W_0 + W_1X_1$	$W_0 + W_1X_1 + W_2X_2$
4	W_3	W_3	X_2	W_3X_2	-	$W_0 + W_1X_1 + W_2X_2$
5	-	W_3X_2	X_1	$W_3X_1X_2$	$W_0 + W_1X_1 + W_2X_2$	$W_0 + W_1X_1 + W_2X_2 + W_3X_1X_2$
6	W_4	W_4	X_1	W_4X_1	-	$W_0 + W_1X_1 + W_2X_2 + W_3X_1X_2$
7	-	W_4X_i	X_1	$W_4X_1^2$	$W_0+W_1X_1+W_2X_2+W_3X_1X_2$	$W_0 + W_1X_1 + W_2X_2 + W_3X_1X_2 + W_4X_1^2$
8	W_5	W_5	X_2	W_5X_2	-	$W_0 + W_1X_1 + W_2X_2 + W_3X_1X_2 + W_4X_1^2$
9	-	W_5X_2	X_2	$W_5X_2^2$	$W_0 + W_1X_1 + W_2X_2 + W_3X_1X_2 + W_4X_1^2$	$W_0 + W_1X_1 + W_2X_2 + W_3X_1X_2 + W_4X_1^2 + W_5X_2^2$

Left-justification of the output mantissa preserves the accuracy of the element, since the maximum number of significant bits are stored in the memory.

The exponent processor determines the output exponent, provides information for mantissa scaling, and adjusts the output exponent for left-justification of the mantissa.

The exponent processor for floating point computation consists of a bit-parallel adder/subtractor, four bit-parallel memories, and control logic.

The e_i store retains the exponent value for the current (i-th) term of the polynomial, while the e_s store retains the exponent for the number already in the accumulator. The processor loads the scaler registers with the necessary shift information so that each new term may be properly added to the accumulator contents. The processor keeps track of the accumulator exponent. When all terms have been accumulated, the e_s exponent is adjusted for the mantissa left-justification and outputted to the X-Y memory. This approach was elaborated upon in Ref. 11.

4.4.2 MICRO-PROCESSOR BASED SYSTEM

An alternate implementation of the bit parallel processor would be based on a micro-processor as the heart of an element. A separate hardware multiply/accumulate unit would be included within the I/O structure of the micro-processor to reduce the time required for the arithmetic computations.

The architecture of such a system is shown in Fig. 10. The micro sequences required to implement any of the polynomials would be stored in the program memory and the processor could handle all of the required housekeeping functions such as address indexing, subroutines, jumps, etc. The data switching and multiplexing, which is handled with discrete parts in the bit parallel approach, would be controlled within the micro. In order to maintain a balance between complexity and speed the system would be limited to 16 bit precision.

A great variation in system speed, complexity, power, cost, etc. exists and the exact factors will depend upon the type of micro employed in the system as elaborated upon in Section VI. The processor could vary from one of the bit slice processor to one of the many 8 bit processors.

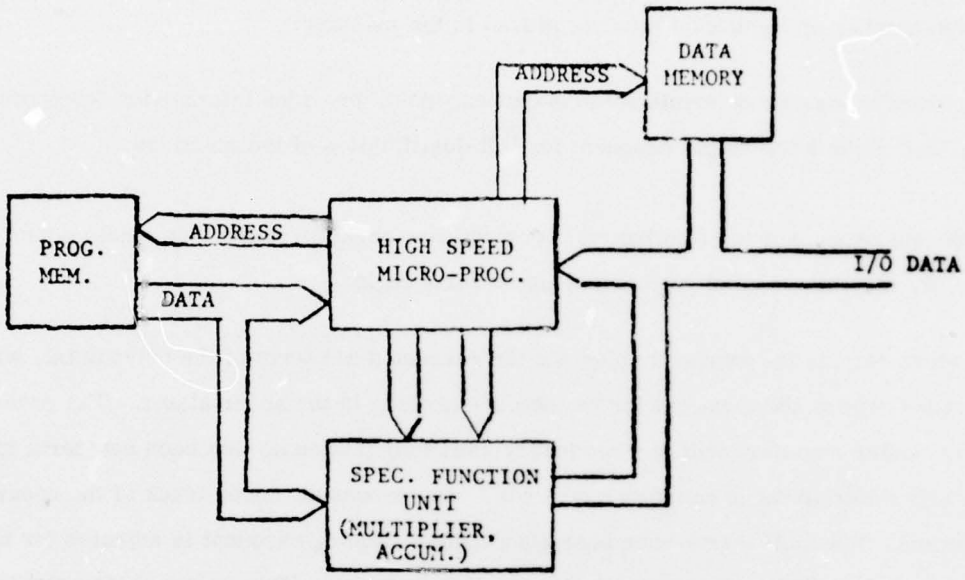


Fig. 19. Micro-Processor Based Element

The bit slice processors offer higher throughput (typical cycle time 100-300 ns) at the expense of system complexity. Most bit slice processors are microprogrammable and require a 16 bit wide control memory. Some of the bit slice processors that could be employed in the system are: the AMD 2900 series, 4 bit bipolar slice; the MMI 5701 series, 4 bit bipolar slice or RCA's ATMAC, 8 bit SOS array.

At the other end of the spectrum are the low cost 8 bit processors which have a cycle time in the order of 2 - 5μs. These devices have a fixed instruction set and are generally simpler to implement. In order to obtain a useful system, double precision arithmetic operations would have to be used. Some typical devices are: the 8080 series, 6800 series both NMOS devices or the 1802 series which is a CMOS device.

4.5 CPA CONTROL SOFTWARE

This section describes the processor (minicomputer or microprocessor) interface structure(s) for control of a CPA. The computer must provide the array with sufficient information (in the form of control instructions and data) to solve the selected problem. Control

instruction and data must be loaded into their respective memories in the proper format since much time can be saved in the actual processing of the data if it emerges from the memory in a set manner.

Three different functions will be described:

1. Array command word structure
2. Element control word structure
3. Data memory organization

Reference is made to the functional diagram shown in Fig. 11. The description of the software interface will be illustrated by a practical example involving an array of five elements to be programmed for a network configuration as shown in Fig. 12.

Figure 13 summarizes the various types of word structures and format presented in the following discussion.

4.5.1 ARRAY COMMAND WORD STRUCTURE

This set of words is used by the controlling minicomputer to instruct the array about what sort of data is to be given to or taken from its memory, and when to start, interrupt and continue array execution. A separate 16 line bus is provided for these commands.

There are three basic types of commands: (load/retrieve data, execute, interrupt). The first is broken down into three subgroups:

1) Data transfer, single.

   This is a single word command to either load or retrieve a single data word from the address specified in the data or control memory. Bits 0 to 9 refer to the address, bits 10 and 11 are zero to indicate a single transfer without address extensions, bit 12 identifies either the control or data memory and bits 13, 14 and 15 form a three bit operation code (2_8 for load, 12_8 for retrieve*)

*The convention used here to read octal digits starting from the 0 bit position (see top of Figure 13).

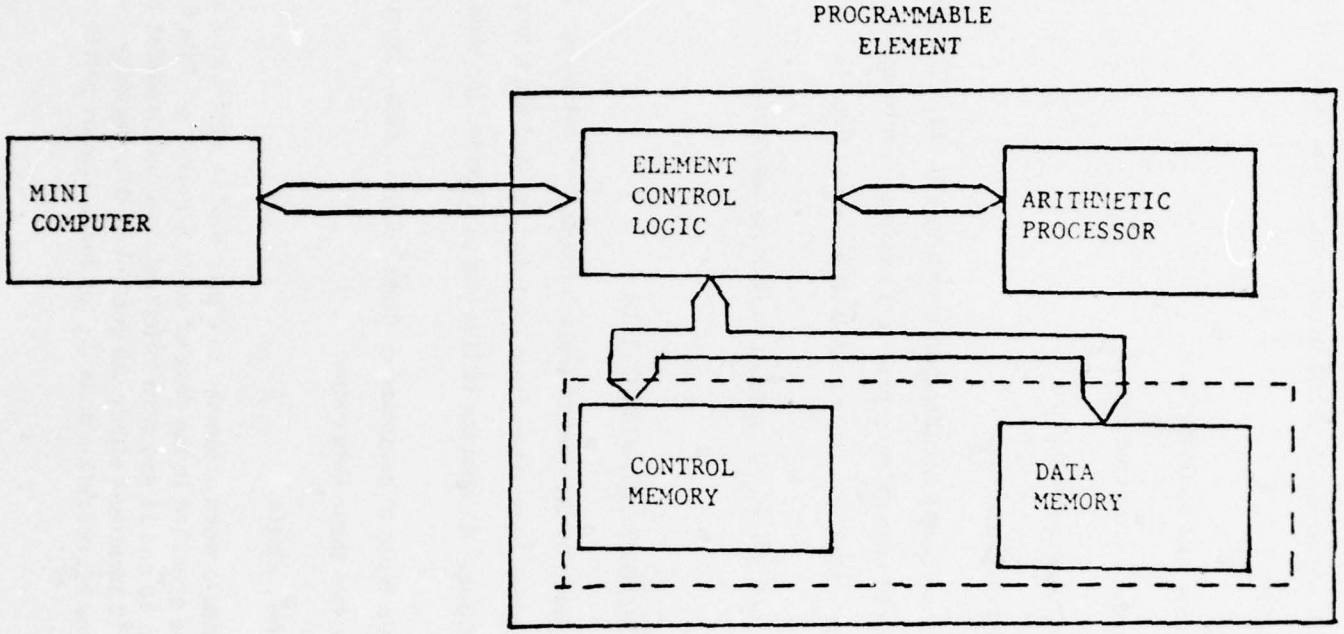
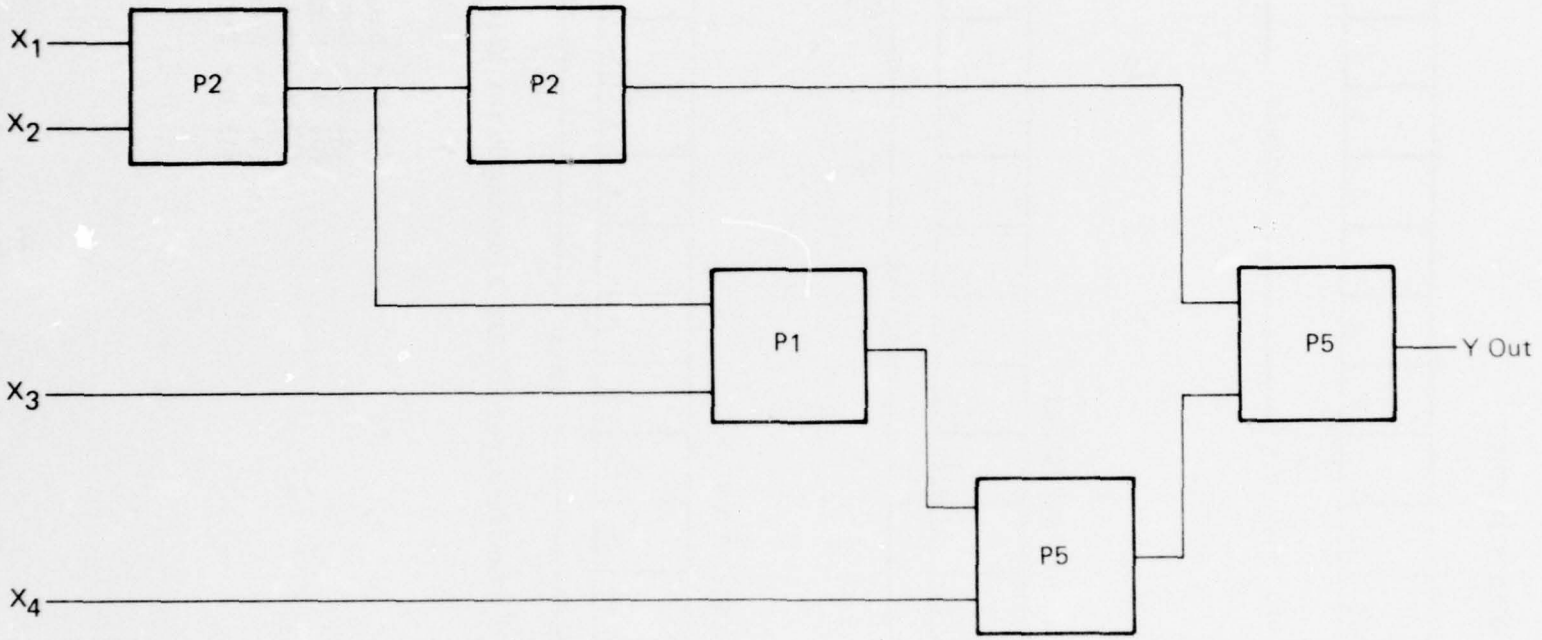
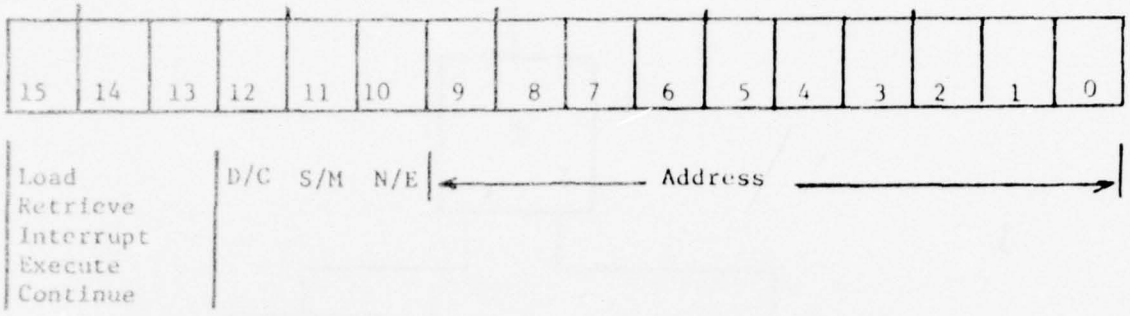


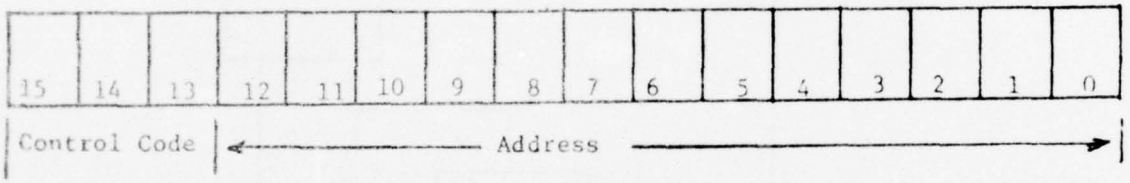
Fig. 11. System Functional Diagram


X1
X2
X3
X4
P2
P2
P1
P5
P5
Y Out

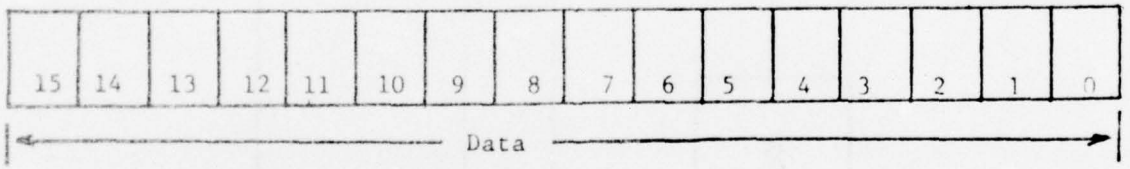
ARRAY COMMAND WORD STRUCTURE



ELEMENT CONTROL WORD STRUCTURE



DATA MEMORY ORGANIZATION



2 Transfers per 32 Bit Floating Point Word

(0) D = Data Memory
(1) C = Control Memory
(0) S = Single Transfer
(1) M = Multiple Transfer
(0) N = Normal (≤ 10 bit) Add.
(1) E = Extended (> 10 bit Add)

Fig. 13. Control and Data Word Formats

2) Data transfer, multiple.

This command uses two words: the first is the same as above except that bit 11 is set to a "1" and indicates that more than one transfer is to take place; the second word indicates the number of transfers.

3) Extended memory.

This command is used when there is more memory in the array than can be addressed by 10 bits (1024) and uses two or three words. The first word is functionally the same as the single word in 1) above, except that bit 10 is set to "1" and bits 0-9 are ignored. If bit 11 is a "0" (two-word command), the second word represents the extended address to which the data will be loaded. If bit 11 is a "1" (three-word command), the second word represents the number of transfers to be made, and the third represents the extended address (up to 16 bits).

The execute command word is used to initiate execution at a specific location in the control memory. Here bits 13-15 are set to "1" (16_8) and bits 12-0 represent the address at which to start execution. Up to 13 bits are available for the address so no provisions will be made for an extended address form of this command.

Finally, the interrupt command word is used when for some reason it is desired to stop execution at some place other than a programmed halt. Provisions are also made for continuing execution from the interruption point should the need arise. The use of this command word is anticipated to be very effective in debugging operations. Bits 15-13 again form the operation code (04_8 for interrupt and 14_8 for continue) with bits 12-0 being unused at present, but are available for future enhancements.

The complete set of command word op-codes is given below:

COMMAND WORD OP-CODES

FUNCTION	CODE BITS 15-13	CODE OCTAL
Spare	000	00
Load	001	02
Interrupt	010	04

Spare	011	06
Spare	100	10
Retrieve	101	12
Continue	110	14
Execute	111	16

4.5.2 ELEMENT CONTROL WORD STRUCTURE

To provide the array with a means of knowing what to do with all the data stored in its data memory, control words are provided in the control memory. These control words are 16 bits long and consist of a three-bit operation code (again bits 13-15) and an address of 10 to 13 bits. The operation code specifies which of five polynomials is to be computed (and some other instructions), while the address tells the processor the location of the first value in the data memory. The various codes are summarized below:

ELEMENT CONTROL OP-CODES

FUNCTION	CODE BITS 15-13	OCTAL
NOP	0 0 0	00
P1	0 0 1	02
P2	0 1 0	04
P3	0 1 1	06
P4	1 0 0	10
P5	1 0 1	12
YDEST	1 1 0	14
HALT	1 1 1	16

The HALT code (16_8) tells the processor to finish off what it was doing and stop processing data. This instruction is the normal way in which a series of computations would end. The code NOP (00_8) is used in special cases when computations overlap and it is desired to use the results of one in the very next computation. This occurs during the division algorithm where a series of P5 polynomials are calculated and each successive computation uses the

results of the previous one. Finally, the YDEST code specifies where in the data memory the result of the previous computation is to be stored. Multiple destinations are indicated by more than one YDEST appearing after a polynomial specification. Shown below is an illustration of a short program corresponding to the network shown in Fig. 12.

LOCATION	CONTENTS	COMMENT
0000	0400001	P2 with data starting at 1_8
0001	1400011	Put result at location 11_8
0002	1400020	Also put result at location 20_8
0003	0400007	P2 with data starting at 7_8
0004	1400027	Put result at location 27_8
0005	0200012	P1 with data starting at 12_8
0006	1400024	Put result at location 24_8
0007	1010022	P5 with data starting at 22_8
0010	1400026	Put result at location 26_8
0011	1010025	P5 with data starting at 25_8
0012	1400030	Put result at location 30_8
0013	1600000	Halt

4.5.3 DATA MEMORY ORGANIZATION

The organization of the data memory is rigidly structured in order to facilitate the task of the controller when accessing data for a particular computation. The polynomial weights are loaded in reverse order, (in ascending address corresponding to descending weight index) as are the X values. Table 5 shows how the weights and X values are loaded. Up to eleven locations are required (for P5) to hold the data for each computation. Since the controller is capable of depositing the result of any calculation into any address in the data memory, dummy values or zeros must be loaded if a particular value is not available until after computation has begun. Note that the control words above instruct the controller as to which polynomial to compute and which data values to expect and in which order.

TABLE 5. DATA MEMORY ORGANIZATION

WORD	CONTENTS	USED IN POLYNOMIALS
1	W_5	P1
2	W_4	P4, P1
3	W_3	P4, P2, P1
4	W_2	P4, P3, P2, P1
5	W_1	P4, P3, P2, P1
6	W_0	P5, P4, P3, P2, P1
7	X_4	P4
8	X_3	P4
9	X_2	P5, P4, P3, P2, P1
10	X_1	P5, P4, P3, P2, P1

A graphic representation of the array's memory content for the specific example considered is shown in Fig. 14. Note that the instructions loaded into the control memory are taken from the previous program, while the data memory contains the weight and variable inputs for each of the polynomials to be computed. These are each 32-bit words (1 bit sign, 23 bit mantissa, 7 bit exponent, 1 bit exponent sign) and correspond to the same format as used in FORTRAN and BASIC languages. The values in parenthesis represent the 4 inputs to the simulated network while the result (Y out) is finally stored in location 30.

In conclusion, to program the array for the computation of the polynominal network shown in Fig. 12 and then instruct the array to execute such computation, the computer will give the array controller the following command sequence:

1) 0/011/100/000/000/001 = 034001 Load Control Memory start at 1
2) 0/000/000/000/001/100 = 00014 with 14_8 words
3) 0/010/100/000/000/001 = 024001 Load Data Memory Start at 1
4) 0/000/000/000/011/000 = 000030 With 30_8 words
5) 1/100/000/000/000/001 = 160001 Execute Starting at 1
6) 1/010/000/000/011/000 = 120030 Retrieve 1 Word from Location 30

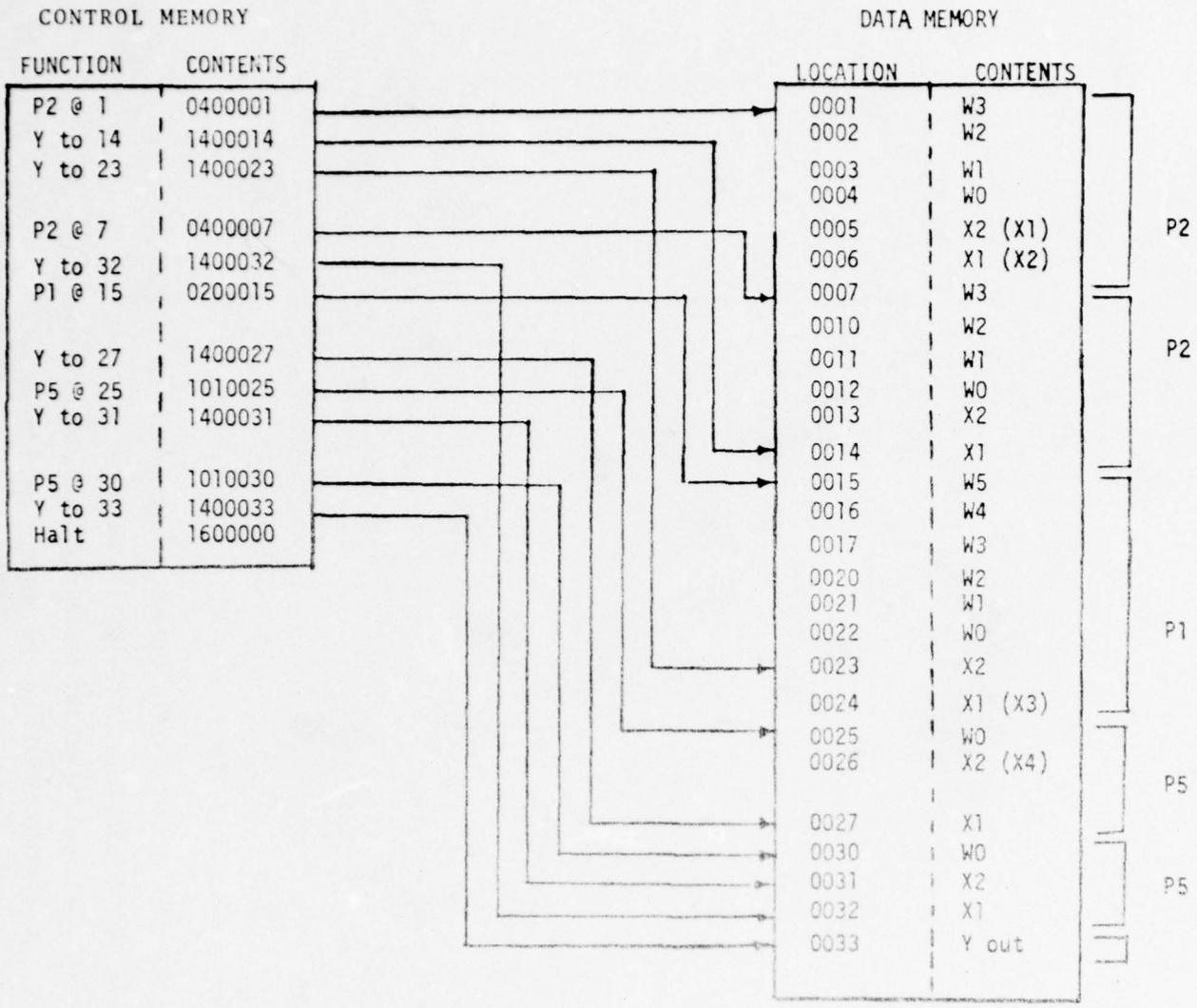


Fig. 14. Typical Array's Memory Content

Note that between commands 2 and 3, and 4 and 5 the actual data transfer takes place over the data bus.

SECTION V

BRASSBOARD ELEMENT

This section describes, first as an overview, and then, in more detail the design, construction and test of the brassboard element. This brassboard was based on the serial type organization with 32-bit floating point precision and was provided with sufficient features to demonstrate the overall element approach, using a group of the custom serial-parallel multipliers, as well as "calibrate" the performance of this architecture against alternate approaches in the tradeoff analysis. While the heart of the element, the 24-bit mantissa processor, incorporating internal switching for function control, did not present undo problems, the exponent processor and associated scalers for justification was found to require a considerable amount of circuit packages based on existing available parts. Much of the other complexity was due to the overall timing and control which used TTL parts for flexibility in the initial design. Any future model would use PROM's for considerable savings in parts.

The actual brassboard is shown in Fig. 15, with the mantissa processor card shown out of the rack. The eight 24-bit CMOS/SOS (ax+b) chips can be seen.

5.1 OVERVIEW

The CPA brassboard (block diagram shown in Fig. 16) has the capability to evaluate either a single 6 term polynomial or 2, 4-term polynomial expressions with 32 bit floating point precision (24 bit mantissa, 8 bit exponent) and was designed to interface with a HP21MX mini-computer. Beside having the capability to evaluate the two polynomials, the control logic and multiplexing circuits necessary to implement a 4 input, 4 output FFT butterfly, transversal filter and the recursive division algorithm are also included within the element as discussed previously. (The addition of two additional output registers are required for the butterfly and an iteration counter and associated logic will be required for the division algorithm).

Since the breadboard does not have a self-contained memory, all operating parameters (poly. type, input data x's and coefficients) are stored in the host processor and transferred

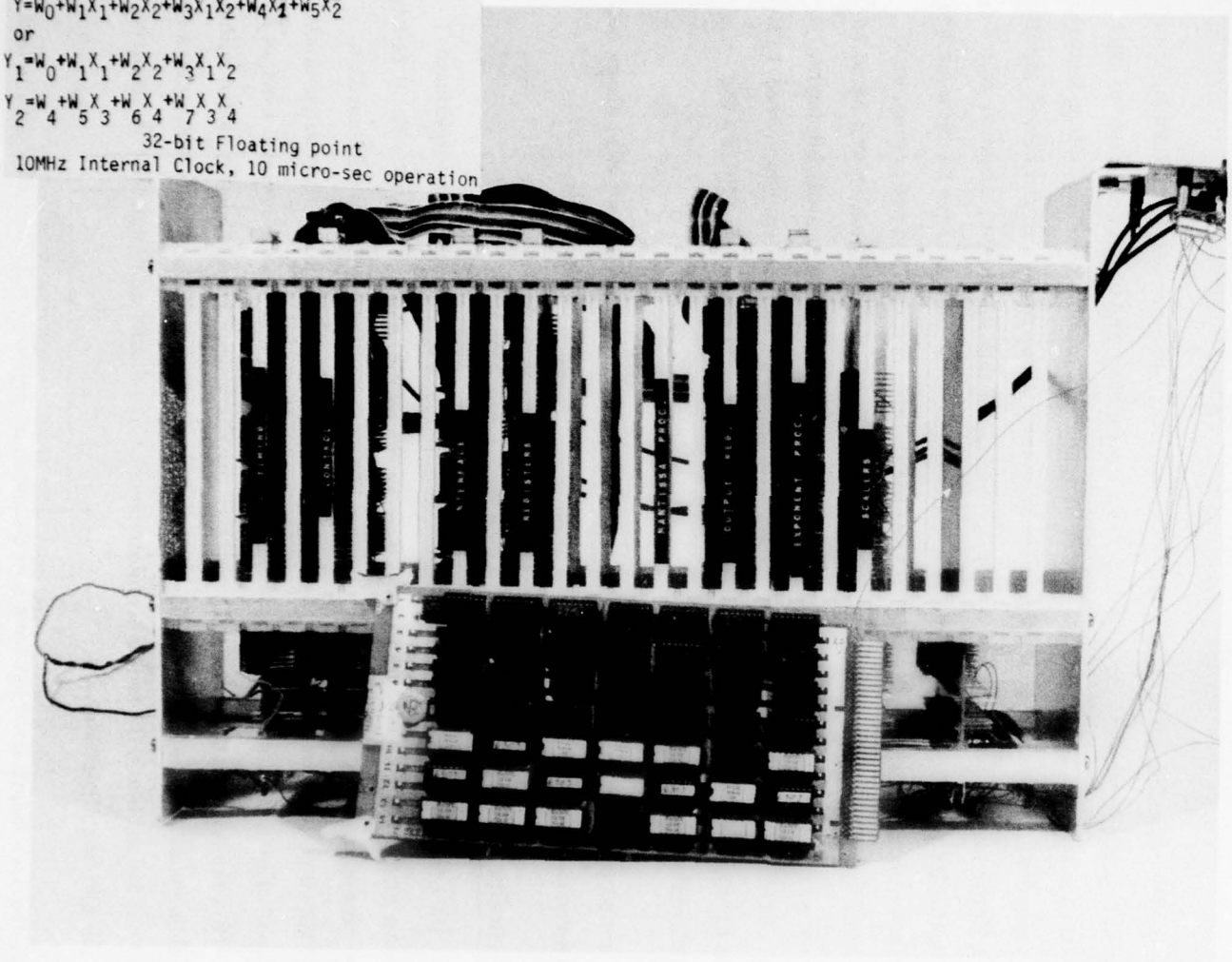


Fig. 15. Brassboard Element

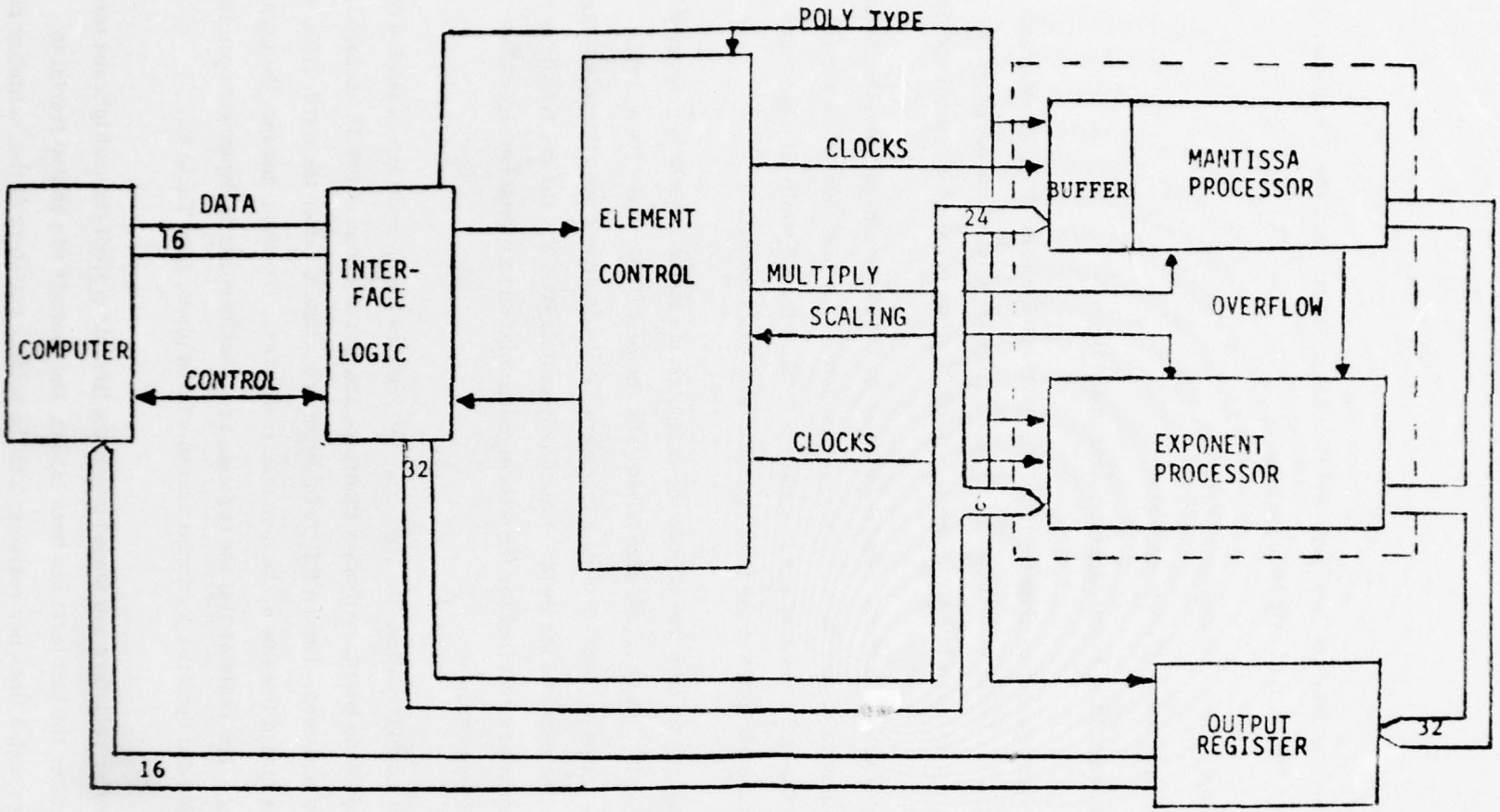


Fig. 16. Brassboard Element Block Diagram

to the CPA element as they are required, and the CPA transfers back the Y's after each implementation. Two sets of data buses are used:

1) 16 bit input bus - computer to CPA

2) 16 bit output bus - CPA to computer

along with two control lines to indicate the status of the bus.

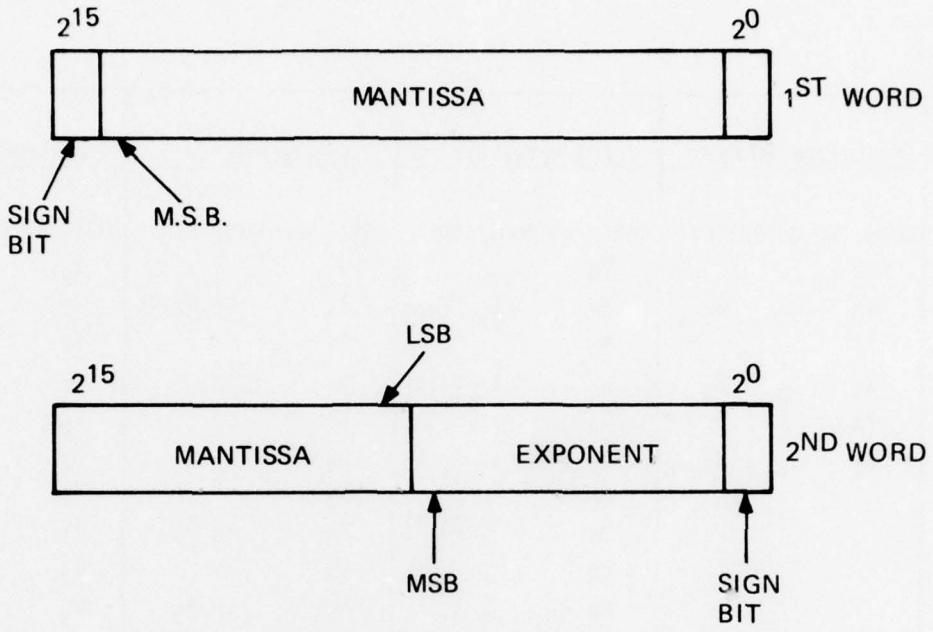
Three different formats are used on the bus, (Fig. 17); one for control information and the remaining two for the 32 bit data word. The bus from the CPA to the computer is used for data transfer only while the computer to the CPA bus is used for control and data.

The time, or number of transfers, required to load the CPA is dependent upon the function that is to be implemented and varies from a minimum of 7 to a maximum of 25 word transfers (see Table 6). All even numbers transfers are data 1 format while all odd number transfers, with the exception of the first transfer, are data 2 format.

In order to transfer data from the CPA to the computer, an output request is generated by the computer, the 2^{14} bit is raised high followed by the poly type. The CPA will then respond with the required number of data transfers, again dependent upon the poly type. Four transfers are required for the 2 - 4 term polynomials (two Y's and two transfers per word), with 2 transfers required for the 6 term polynomial and the division algorithm. The FFT requires 8 transfers.

During the input loading routine, the first transfer contains the op-code and is latched on the computer interface board. All other transfers are stored in one of two 16 bit parallel-in, parallel-out registers. Once a full 32 bit word is available, (after the third, fifth, etc. transfers), it is loaded into one of the nine buffer registers, (24 bits), and into the exponent (8 bits) memory. The register that the contents are transferred to is dependent upon the poly type decoded and upon the particular transfer taken place, (see Table 6).

Actually, the buffer registers are loaded twice; the first time all of the multiplicand terms are loaded and after the last term has been loaded, the contents are shifted (serial to parallel transfer with 3 lines per register) into the holding registers of the multiplier chips.



FORMAT DATA WORD

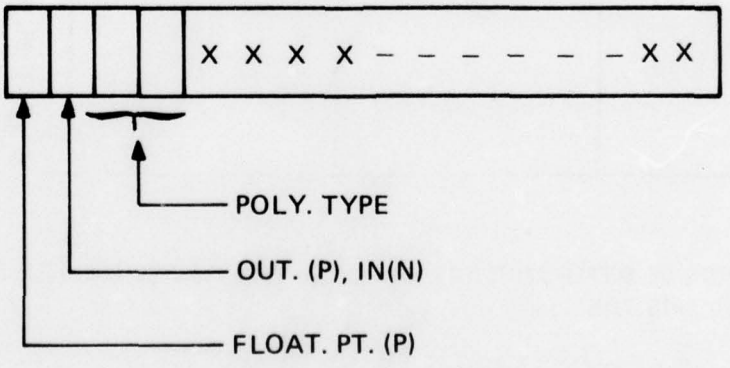


Fig. 17. Format Control Word

TABLE 6. CPA LOADING

CPU TRAN.	DIVISION ALGO.	FFT BUTTERFLY	6 TERM POLY.	4 TERM POLY.
1	CONT. WD 0000/1000	CONT. WD 0001/1001	CONT. WD 0010/1010	CONT. WD 0011/1011
2	W_0	W_R	X_2	X_2
3	W_0 L_D B_4	W_R L_D B_2,B_5	X_2 L_D B_2,B_5	X_2 L_D B_2
4	X_1	W_i	X_1	X_1
5	X_1 L_D B_5	W_i L_D B_1,B_2	X_1 L_D $B_{1,3,4,6}$ *	X_1 L_D $B_{1,3,6}$
6	X_2 *	Q_R *	W_5	X_2'
7	+ X_2 L_D B_3,B_4	Q_R L_D B_2,B_4	W_5 L_D B_5	X_2' L_D B_5
8		Q_i	W_4	X_1'
9		Q_i L_D B_1,B_5	W_4 L_D B_4	X_1' L_D $B_{4,6}$ *
10		P_R	W_3	W_3
11		P_R L_D B_7	W_3 L_D B_3	W_3 L_D B_3
12		P_i	W_2	W_2
13		+ P_i L_D B_8	W_2 L_D B_2	W_2 L_D B_2
14			W_1	W_1
15			W_1 L_D B_1	W_1 L_D B_1
16			W_0	W_0
17			+ W_0 L_D B_7	W_0 L_D B_7
18				W_3'
19				W_3' L_D B_4
20				W_2'
21				W_2' L_D B_5
22				W_1'
23				W_1' L_D B_6
24				W_0'
25				+ W_0' L_D B_8

* CONTENTS OF BUFFER REGISTERS ARE LOADED INTO HOLDING REGISTER IN MULTIPLIERS DURING THIS TIME

+ EXECUTION OF PROBLEM START AT END OF TRANSFER

The registers are then reloaded with the multiplier inputs and the constant (W_0, etc.) terms. The contents are then serially shifted into the processor as required.

The switch point between multiplier and multiplicand terms is dependent upon poly type and is indicated by a "*" in Table 6.

5.2 MANTISSA PROCESSING

The mantissa portion of the element is organized as a word parallel-bit serial processor (as discussed previously) that can be re-configured to perform any one of the family of multinomials.

The operation of the processor is broken into five sections:

1) loading of multiplicand into the holding register located within the multiplier chip. (occurs during transfer cycle between CPU to the element).

2) first level multiplication - for all single product terms (W_1X_1, etc.)

3) second level multiplication - for all cross product or squared terms ($W_4X_1^2$, etc.)

4) scaling operation

5) final multiplication and summation

A timing budget is shown in Fig. 18.

5.3 EXPONENT PROCESSING

The exponent processor, unlike the mantissa processor, operates in a word serial, bit parallel fashion for maximum compatibility with the mantissa processor and is programmable to operate on any one of the family of multinomials as described in Section IV.

The combination of mantissa and exponent processor constitute the floating-point arithmetic unit of the element. As the mantissa processor multiplies the mantissa of two numbers the exponent processor adds the corresponding two exponents to determine the exponent of the mantissa product. If two or more multiplications are performed in parallel and the results added together (as in the case of a polynomial evaluation), all input data must be represented with mantissa/exponent normalized to the largest of the exponents. Furthermore, each

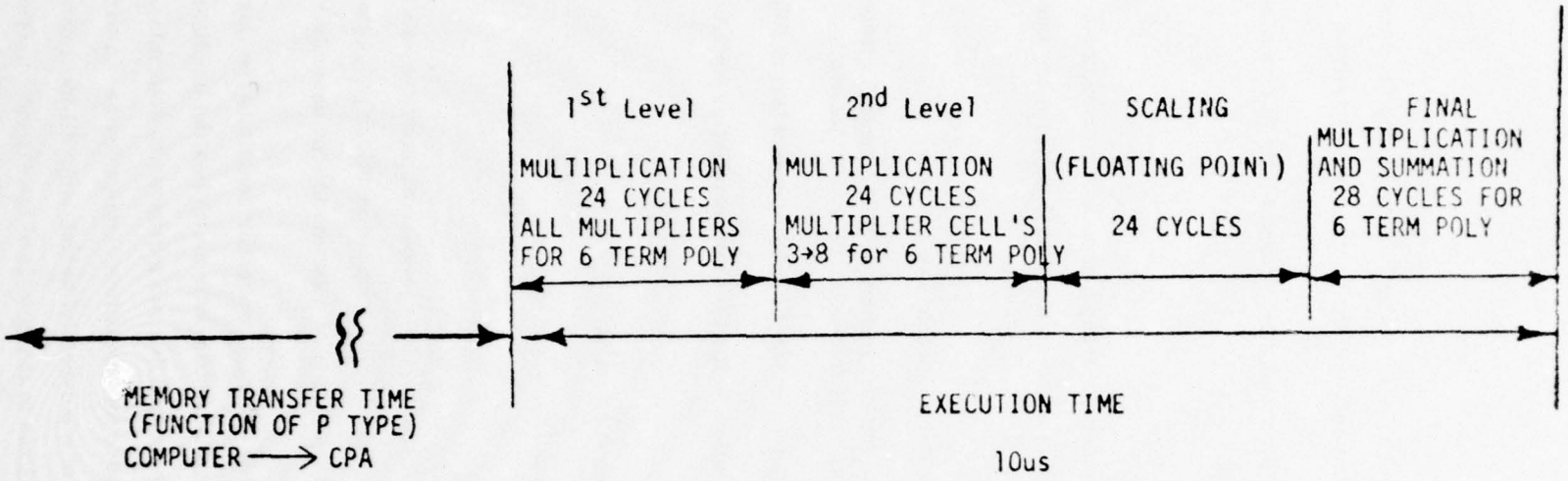


Fig. 18. Timing Budget for CPA Element

partial sum of two products has to be checked for possible overflow. Should overflow occur the corresponding exponent has to be scaled accordingly.

The output register board performs the left justification on the mantissa, transferring the number of overflow/underflow bit positions back to the exponent processor. When an output command is generated by the computer, the register board selects the first 16 MSB's of the mantissa and gates them on to the output bus and then selects the 8 LSB's along with the 8 bit exponent word for transfer back to the computer. The handshaking commands for the I/O bus are generated by the register board and combined with the gating located on the computer interface board to control the transfers.

5.4 DESCRIPTION OF LOGIC DESIGN

The basic brassboard element was built around the CMOS/SOS serial-parallel multiplier (TCS 039) with all of the control and data handling logic, (except for the exponent processor), using standard TTL parts. The element required 8 wire-wrap boards (4.5 "X 9") used about 200 IC's and consumed 30 watts.

The board breakdown is:

1) timing
2) control
3) CPA interface (to HP 21MX)
4) buffer register
5) mantissa processor
6) output register
7) exponent processor
8) scalers

5.4.1 TIMING AND CONTROL

These boards control the mantissa portion of the CPA and were built using discrete TTL IC's in order to maintain maximum flexibility regarding bit size, poly type and numbering system (fixed point of floating point).

The timing chain consists of a 115 bit serial shift register which is held in the all zero state until the data transfer between the computer and the element has been completed. After the last data word has been stored, a "start" command is generated by the interface logic. This input is applied as the serial input to the register and is clocked into the first stage. The "start" signal returns to a zero and the "one" in the register is shifted thru all the stages with various outputs being applied to the control board.

Beside the start pulse, the timing board is reset by the interface logic prior to any operations and also receives the 4 bit op-code which is decoded and latched on the interface board.

The op-codes used to control the multiplexers which by-pass sections of the timing chain are as follows:

a) single or two level multiplications - the FFT or transversal filter (P3 or P4) requires only a single level multiplier while all other operations require two cascaded multipliers; therefore, the 24 stages that control the operations of the second level multipliers can be eliminated during the execution of the FFT.

b) fixed or floating point - for fixed point systems the 24 stages required for the scaling operation is bypassed.

c) combination of the above selects single level multiplication with fixed point operation.

The implementation of the multiplexed switching for this timing and control chain is shown in Fig. 19, with the execution time for the various conditions shown in Fig. 20. Whenever the timing chain is broken, a zero is inserted into the unused portions, thus preventing any "1's" from entering the unused portion of the registers which could generate false control signals.

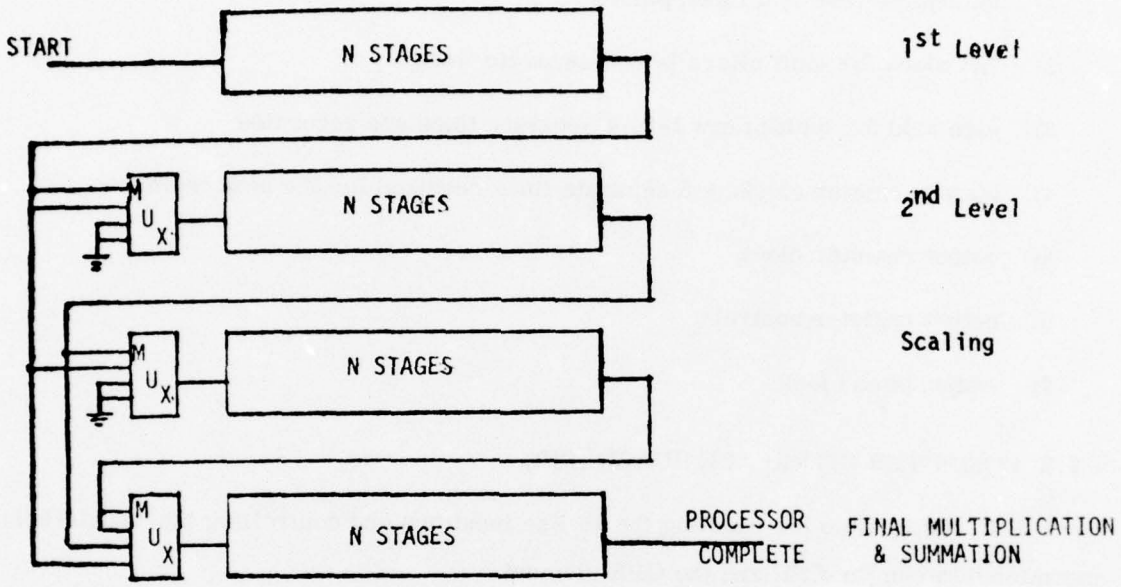


Fig. 19. Configuration of Timing Chain

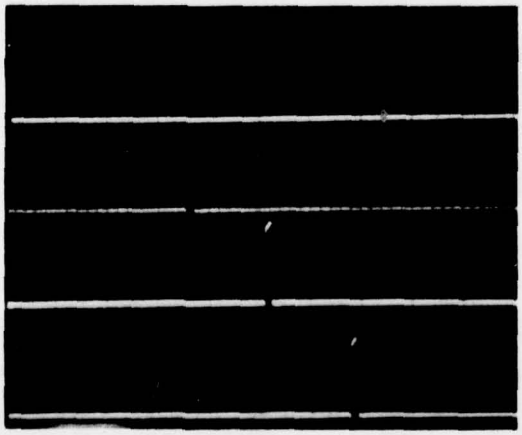


Fig. 20. Execution Times vs. Poly Type

The control board combines the various outputs of the timing chain and generates system clocks and controls. The following lines are generated:

1) multiplier reset - 2 clock pulses
2) "p" clock for multipliers 1-8, 7 separate lines
3) sign hold for multipliers 1-8, 4 separate lines are generated
4) buffer register clocks - 5 separate lines required for the nine registers
5) output register clock
6) output register control
7) adder/latch clock

5.4.2 COMPUTER INTERFACE BOARD (CIB)

The CIB is responsible for handling the 16 line input bus and controlling the "handshaking" operation between the CPU and the CPA element.

The major functional blocks on the CIB are:

1) transfer counter and decode logic
2) op-code latch and decode logic
3) serial to parallel converter - 2-16 bit bus to single 32 bit bus
4) multiplexer logic to load buffer registers
5) burst counter - load multiplier
6) handshaking logic
7) miscellaneous logic to control various timing pulses

Initially, the transfer counter is held in the zero state, either by applying an external master reset or by the internal reset generated at the completion of a cycle, and the CPA flag is in the high state (not busy). When the CPU has data available on its bus, it raises the control line thus signaling the interface board that data is available. The transfer counter will advance to state one and generate a strobe which will latch the data

(op-code if it's the first transfer). Once the op-code has been latched, the multiplexers will be set according to the "P" type and thus determine the sequence for loading the buffer registers. It will also determine when to reset the transfer counter and generate the start pulse. See Fig. 21 for a load example for the 6 term polynomial.

5.4.3 BUFFER REGISTER BOARD

The register board serves as the intermediate memory required between the host computer and the mantissa processor of the CPA. It consists of 8-24 bit parallel to serial registers, and 6 circuits to perform a 1's complement function.

Since all the numbers generated by the computer are in 2's complement form and since the TCS 039, serial/parallel multiplier requires that the parallel word be a sign-magnitude form, any negative number must be converted. Since the multiplicand, the parallel word in the multiplier, is loaded in a serial-parallel fashion, (3 lines with 8 bits per line), a one's complement is performed on the negative numbers as they are being loaded into the registers. This is accomplished by latching the sign-bit in an external register and using the register to control one input of an exclusive-or gate. Therefore, for negative numbers the output is inverted while for positive numbers on inversion takes place.

This method, while being simple to implement, does distract from the system accuracy since it introduces an error of one bit in all negative numbers. In the future, it is recommended that a parallel 2's complementor, with necessary control logic, be placed between the interface board bus and the buffer register board.

The loading sequence of the registers is controlled by the interface board and is dependent upon the operation being performed. See Table 7 for the loading sequence as a function of "P" type.

5.4.4 MANTISSA PROCESSOR BOARD

The processor board is built around the custom SOS/CMOS serial/parallel multiplier, TCS 039, and is implemented in a word parallel, bit serial approach as discussed previously.

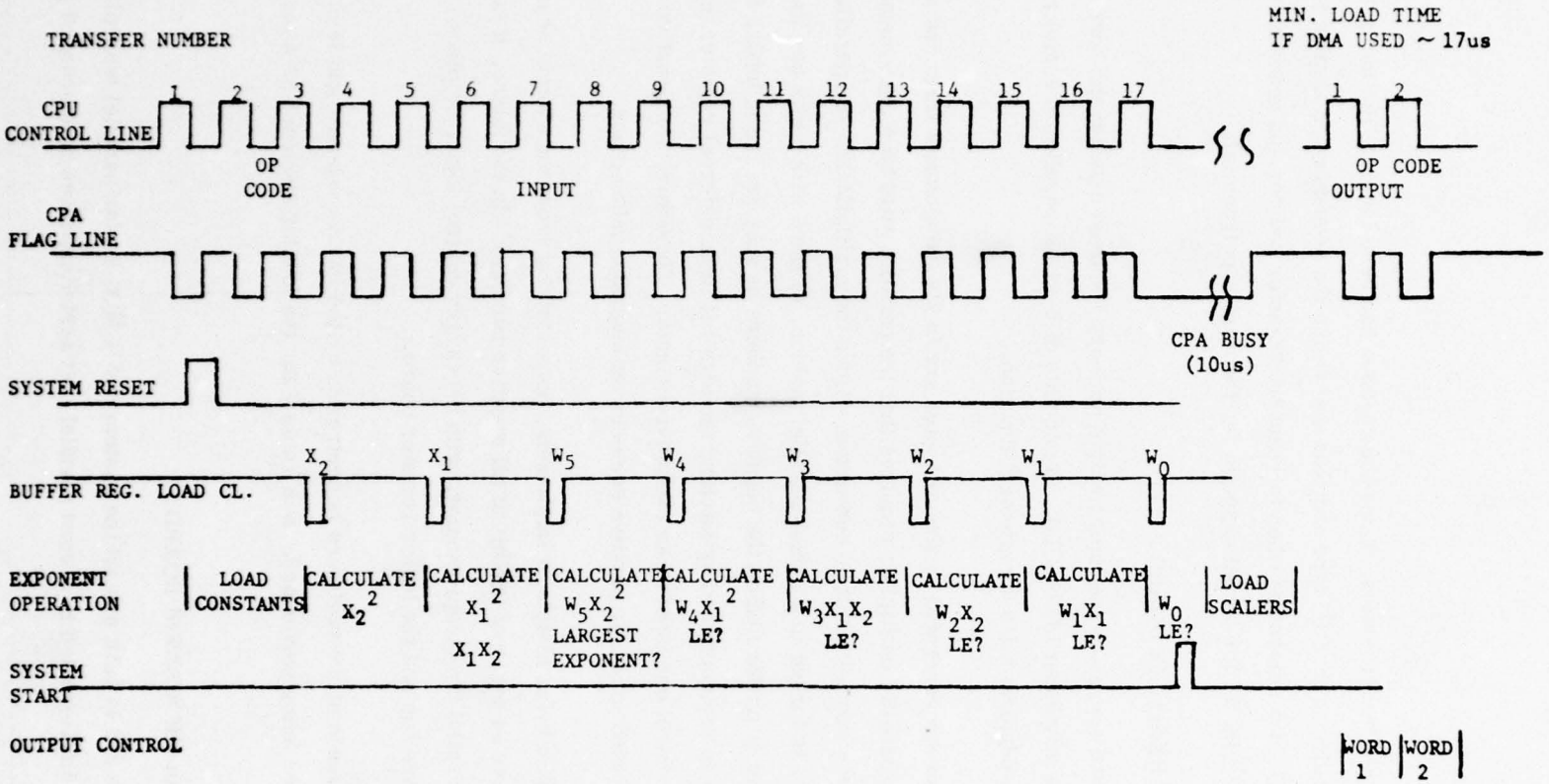


Fig. 21. Load Example (Computer CPA Timing Diagram For 6 Term Polynomial)

TABLE 7. MULTIPLIER-ADDER FUNCTIONS

MULTIPLIER	P_1 6 TERM POLY	P_2 4 TERM POLY	P_3 COMPLEX BUTTERFLY	P_4 FILTER	P_5 DIVISION
1	W_1X_1	W_1X_1	$-W_iQ_i$	W_0S_0	Nu
2	W_2X_2	W_2X_2	W_RQ_R	W_1S_1	W_0X_2
3	W_3X_1	W_3X_1	Nu	Nu	X_1X_2
4	$W_3X_1X_2$	$W_3X_1X_2$	Nu	Nu	$-X_1(X_2)^2$
5	W_4X_1	$W_3'X_1'$	Nu	Nu	Nu
6	$W_4X_1^2$	$W_1'X_1'$	$-W_iQ_R$	W_2S_2	Nu
7	W_5X_2	$W_2'X_2'$	Nu	Nu	Nu
8	$W_5X_2^2$	$W_3'X_1'X_2'$	W_RQ_i	W_3S_3	Nu

ADDER	IN / OUT		IN / OUT		IN / OUT		IN / OUT		IN / OUT	
1	M1, M2	A2	M1, M2	A2	M1. M2	A2, A3	M1, M2	A2	Nu	
2	A1, A4	Y	A1, A3	Y	PR, A1	PR'	A1, A4	Y	Nu	
3	W0, M4	A4	W0, M4	A2	PR, A1	QR'	Nu		M2, M4	Y
4	A3, A6	A2	A5, A6	Y'	Pi, A6	Qi'	Yn-1, A6	A2	Nu	
5	Nu		W0', M7	A4	Pi, A6	Pi'	Nu		Nu	
6	M6, M8	A4	M6, M8	A4	M6, M8	A4, A5	M6, M8	A4	Nu	

There are 8 multipliers required with each multiplier handling two, 24 bit numbers generating a double precision product (48 bits). In order to achieve the flexibility required to configure the network to solve various polynomials, T^2L and CMOS multiplexers are provided.* Once the "steering" logic has been established, the output products of the various multiplier cells are added, serially, and stored in an output register.

The functions capable of being handled by the processor were discussed previously. The methods by which they are generated will be shown in detail. Fig. 22 shows the circuit positions of all mantissa components - multipliers, adders and multiplex switches (for poly, control) interconnected for a P1 while Table 7 shows the functions for each multiplier and adder for each poly type. Fig. 23 shows the waveforms associated with a typical example, as a problem progresses through the mantissa processor.

5.4.5 EXPONENT PROCESSOR BOARD

The exponent processor board operates in a word serial, bit parallel fashion and contains its own controller (PROM) to enable it to operate on any one of the four problem types. The exponent is an 8 bit number and is represented in 2's complement format. The basic operation of the exponent processor is as follows:

1) it receives and stores the ordered sequence of exponents corresponding to the data X's and coefficients W's. It is also instructed about the polynomial type to be evaluated.

2) it generates and stores, as required by the polynomial type, any new exponent of the type X_i^2 or X_iX_j.

3) it compares all exponents, determines the largest and generates the necessary number of scalers, one per multiplier output, to be sent to the mantissa processor.

4) it receives overflow indication from the mantissa, computes the correction and scales the exponent of the result accordingly.

The operation of the exponent actually starts with the first transfer from CPU to the CPA. The sequence of operation will be reviewed with the aid of the timing diagram of Fig. 21, and the exponent block diagram of Fig. 24.

*Depending on whether they appear between two multipliers (operating at 10v) or two TTL adders (operating at 5v).

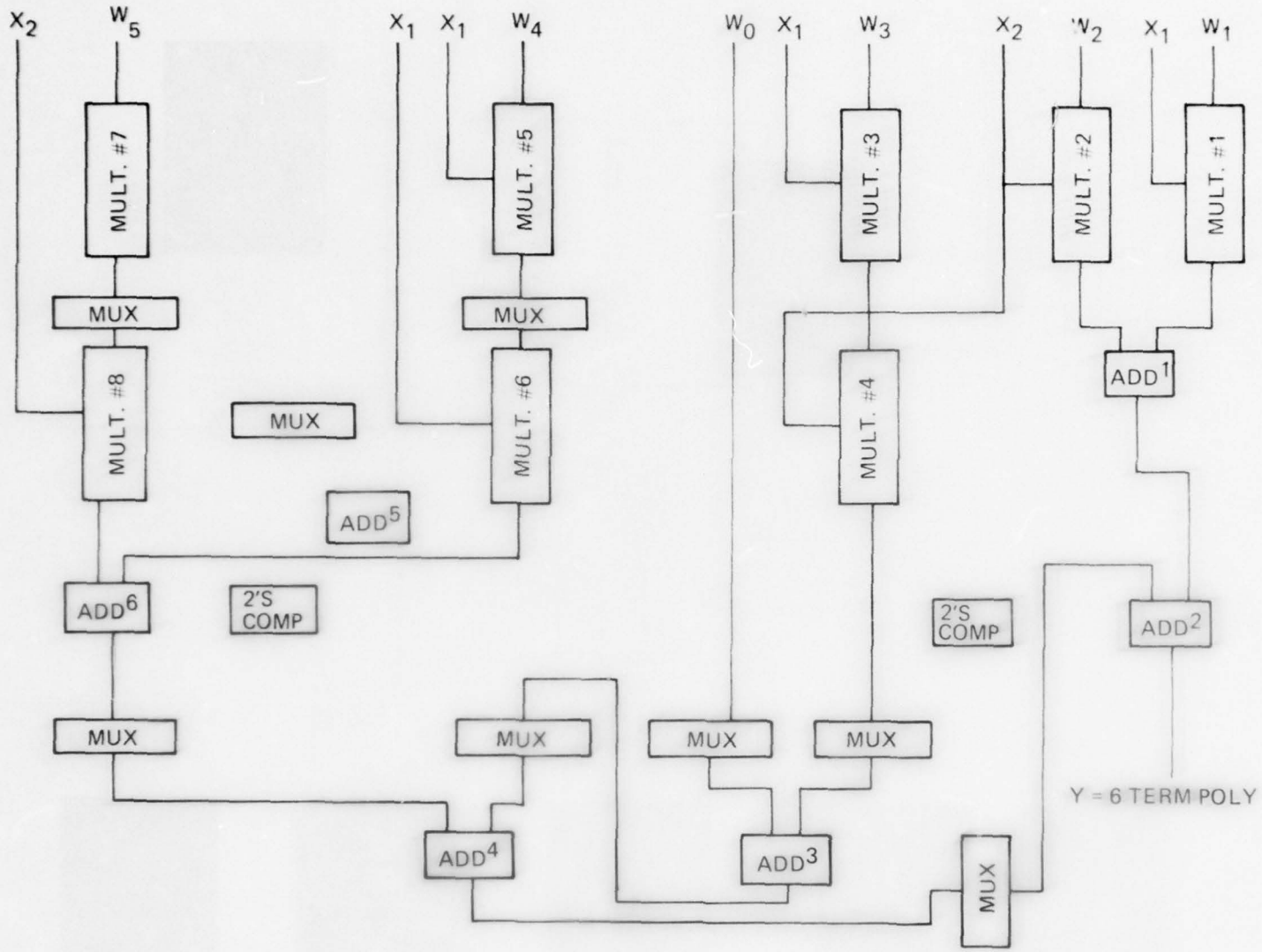


Fig. 22. Basic Mantissa Processor (Shown Intra-Connected For a P1)

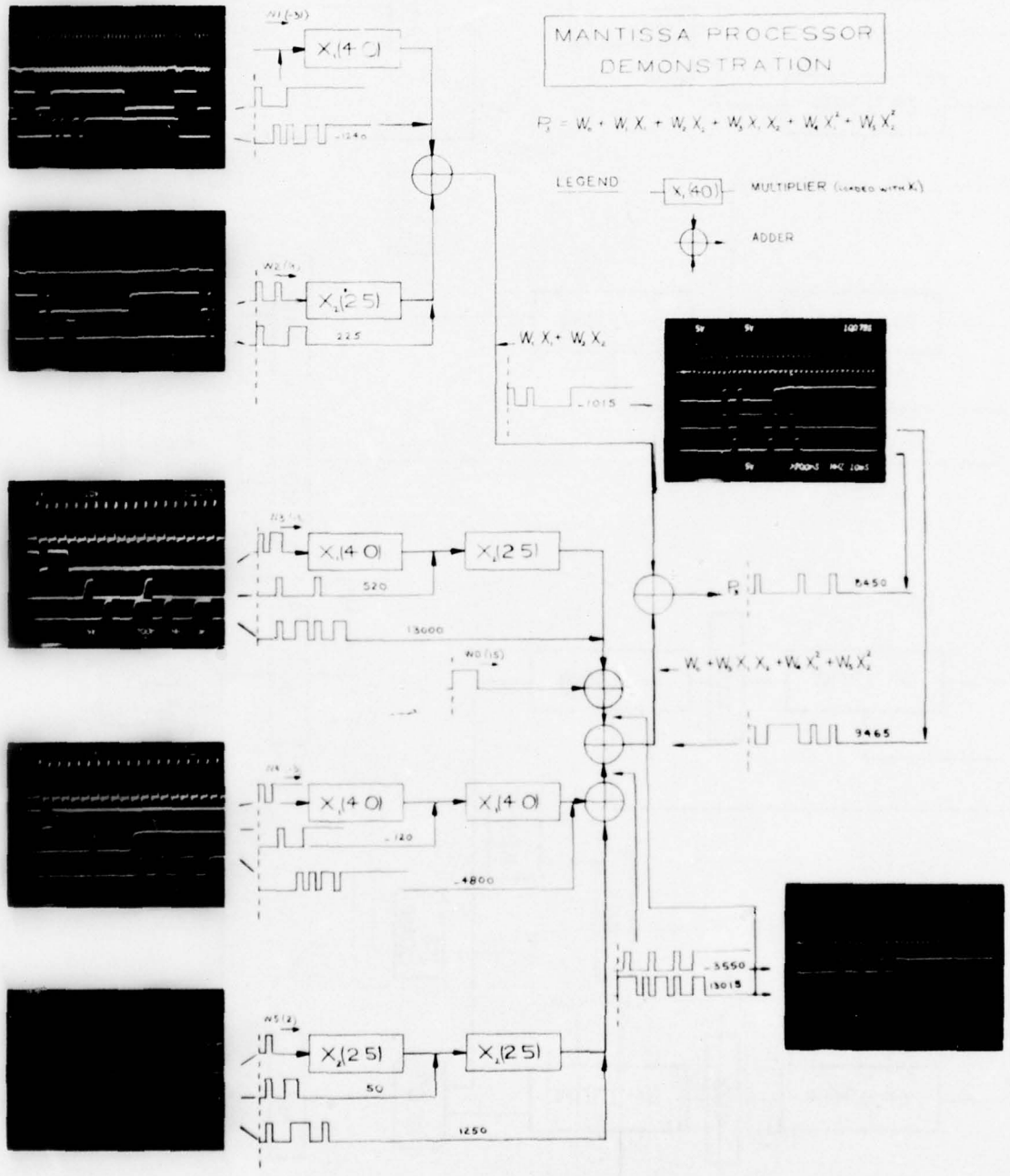


Fig. 23. Mantissa Processor Waveforms

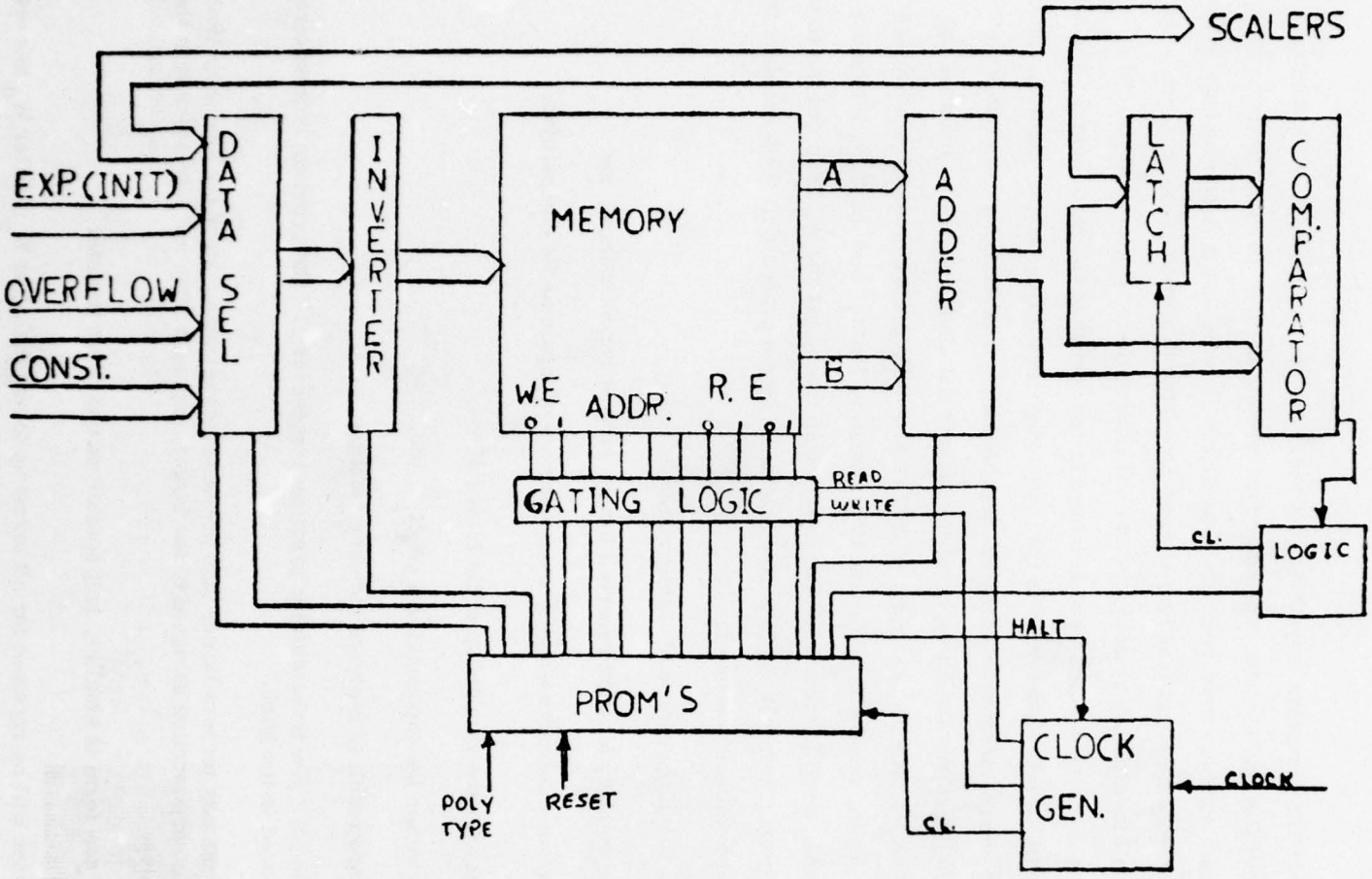


Fig. 24. Exponent Processor

The first transfer generates a system reset which is used to reset the PROM controller and miscellaneous flip flop's on the exponent board as well as other portions of the system. After the reset has been removed, the controller will cycle thru a fixed sequence to generate and store the constants that may be required during the execution of the problem. The controller will then switch to an idle state until instructed to advance. At the end of the third transfer, the X_2 data word is stored in the computer interface board. The word is split up with 24 bits transferred to the buffer registers for use in the mantissa and the remaining 8 bits transferred to the exponent processor. This transfer reinitiates the controller which stores the X_2 exponent in memory and also proceeds to generate the exponent for X_2^2. The X_2^2 exponent is then stored in its proper memory location and the controller, again, turns off. After the fifth transfer, the X_1^2 and the X_1X_2 terms are calculated and stored in memory. The next transfer contains the W_5 term and the first complete exponent term ($W_5X_2^2$) is now calculated by the processor. This result is then stored in three different locations:

1) the location reserved in memory for it,

2) the memory location reserved for the largest exponent term, and

3) the outboard latch which provides one set of inputs to the comparator.

The sequence for the remaining terms is as follows:

1) calculate the exponent term ($W_4X_1^2$, etc.)

2) store result in assigned memory location

3) compare new term with the previous largest term. The previous largest term is stored in the latch.

4A) if the new term is larger than previous largest term, store the new term in the memory location assigned to the largest exponent term and also store it in the latch.

4B) if new term is smaller, halt operations until next transfer.

This sequence will be repeated for all terms up to and including W_0. After W_0 has been processed, instead of the controller halting operation, it will begin a new sequence of instructions.

This sequence will be used to calculate the difference between the largest exponent and all other exponent terms and also to load the difference into the proper scaler. This is accomplished by reading the largest memory location and holding the value in memory output port A. The adder stage is reconfigured to operate as a subtractor and all previous stored exponent terms ($W_5X_2^2$, $W_3X_1X_2$, etc.) are subtracted from the largest term. The output of the subtractor is then loaded into one of the scalers. This operation continues for all terms, and after the last term the controller halts again. The exponent processor will remain idle until the overflow/underflow data is fed back from the mantissa.

The scalers used in the exponent processor consists of pre-settable down counters, and multiplexing logic to control the loading sequence. There are a total of nine scalers, each of which consists of two 4 bit binary counters, with the pre-set inputs being controlled by a common bus connected to the exponent output. The individual load commands are generated on the scaler board and are determined by the "P" type in conjunction with the exponent board.

After being loaded, the scalers are inhibited until a control signal is generated by the timing board. This signal inhibits any multiplier clock pulses from being generated by the control board and turns control of the multiplier over to the scalers. The scaler will provide a variable number of clock pulses to each multiplier cell depending upon the problem being executed. The number of pulses will range from a minimum of zero, for the multiplier cell associated with the largest exponent, to a maximum of 24. Since there are 24 bits being used in the mantissa, this is the maximum number of scaling pulses required to "zero out" any number. Regardless of the number of scaling pulses generated, the mantissa processor control will remain idle until a maximum cycle (24 clock pulses) is complete. After this time it will assume control and generate the required number of clocks to complete the multiplication and addition.

5.4.6 OUTPUT REGISTER BOARD

The output register board performs the following:

1) left justifies the 24 bit mantissa and updates the exponent

2) serial to parallel conversion of the mantissa

3) combines the 8 bit exponent with the 24 bit mantissa and transmits the output words back to the computer.

The present board consists of the following blocks:

1) 2 - 32 bit serial in - parallel out shift registers

2) 2 counters and associated logic to locate the MSB's (MSB detector counters - MC)

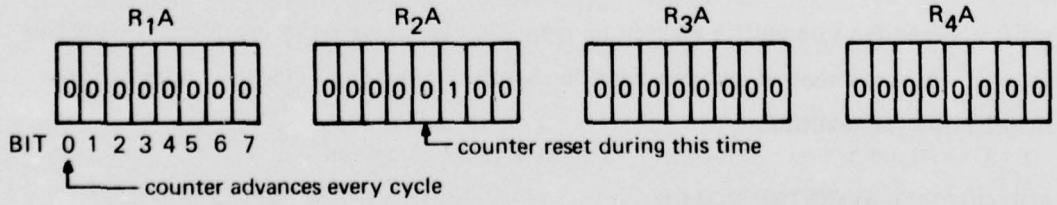
3) PROM controller to control the output select and transfer sequence

4) 4 - 1 multiplexers and latch

By way of describing the operation of the output register, assume that a 6 term polynomial is being executed.

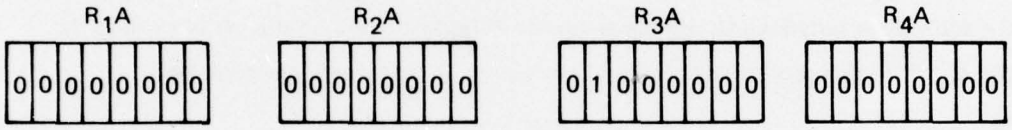
The serial output of the mantissa processor is shifted into a 32 bit shift register. As the data is shifted in, the present bit is compared with the previously received bit and if they are the same, a MSB detector counter is advanced by one; if they are different the MC is reset to zero. The MC is used to locate the MSB of the mantissa, since, depending upon the problem the location of the MSB can vary from 5 bit overflow to 24 bit underflow.

Assume that the two inputs to the final adder stage (adder #2) on the mantissa are +.9000... and -.89690375. The difference between these two terms is: +.00390625 (0.0000000100...). As this number is being shifted into the output register, five extra cycles are provided for possible overflow; therefore, at the end of the processing cycle the following bit pattern is stored in the register:



and the M counter has counted up to 12 ($0\text{-}1100_2$). At this point, the control logic switches the output register enable line and this latches the contents of the M counter (this number is then transmitted to the exponent board for final processing).

In order to simplify the output transfer logic, the sign bit has to be located in bit 0 of either R_1A, R_2A, R_3A or R_4A. As shown by the example, the sign bit and MSB are located in the middle of R_2A. In order to align the bits, the 2 LSB outputs of the M counter is decoded and are used to generate a clock burst. This burst lasts until the 3 LSB's of the M counter are in a zero state, and for the example shown, four pulses will be generated. The M counter will now be at 16 (100000_2) and the registers will be as follows:



The two MSB's of the M counter are decoded in order to locate the register that contains the MSB of the mantissa before the parallel transfer to the computer. The decoding sequence is shown below:

COUNTER MSB		MSB REG.
0	1	R_2A
1	0	R_3A
1	1	R_4A
0	0	R_1A

The following operations are performed:

0) the MSB's of the M counter are jammed into the output multiplexer sequence counter (OMC) and the outputs of this counter are used to control the 4-1 mux.

At this point the cycle is halted until an output request is received from the computer. When the output request is received, a PROM sequence is generated which performs the following:

1) CL1 is generated - latching the 8 MSB's of the mantissa

2) OMC counter is advanced by one, the 4-1 multiplexers outputs are the next 8 MSB's.

3) CL2 is generated - latches 8 bits.

At this point the first 16 bits of the mantissa are available, the ROM sequence is halted and the output word is transferred to the computer.

When the second output request is received the OMC counter is again advanced selecting the 8 LSB's of the mantissa and then the 8 bit exponent is selected and latched. Again, 16 bits are available for transfer and then the operation is completed.

For the four term polynomial, the sequence is repeated twice, the first pass uses M counter 1, while the second pass uses M counter 2; 4 transfers are required.

5.5 SOFTWARE SYSTEM FOR THE BRASSBOARD ELEMENT

Fig. 25 shows the four major parts to be considered in the operational system for the CPA element interface with the HP2108. The overall control is provided by Basic Control System (BCS) supplied by the computer manufacturer. This module handles all I/O operations in order to relieve the user from the task of specifying the I/O slot of each peripheral, thus allowing the user programs to be device independent.

The remaining three modules were written by RCA and are required to control the various data transfers between the element and the computer. The first of these is "D.66" otherwise known as the Drives (listing of all modules are located in Appendix K). Briefly its function is to receive data which has been structured and formatted by the "service routine" and make appropriate calls to BCS in order to deliver and retrieve data from the element. Control information is also provided to the driver in and or to specify how many I/O transfers are to be made to the element. The driver allows the element to operate on an interrupt basis, making it compatible with other peripherals.

The next module is the "Service Routine." This module decodes subroutine calls from the user program to determine which polynomial is desired. From this information, calculations are made to determine the addresses of the data points, how many data points there are to be transferred, and how many results are to be recovered from the element. This and other necessary data are formatted and ordered so as to allow the driver to

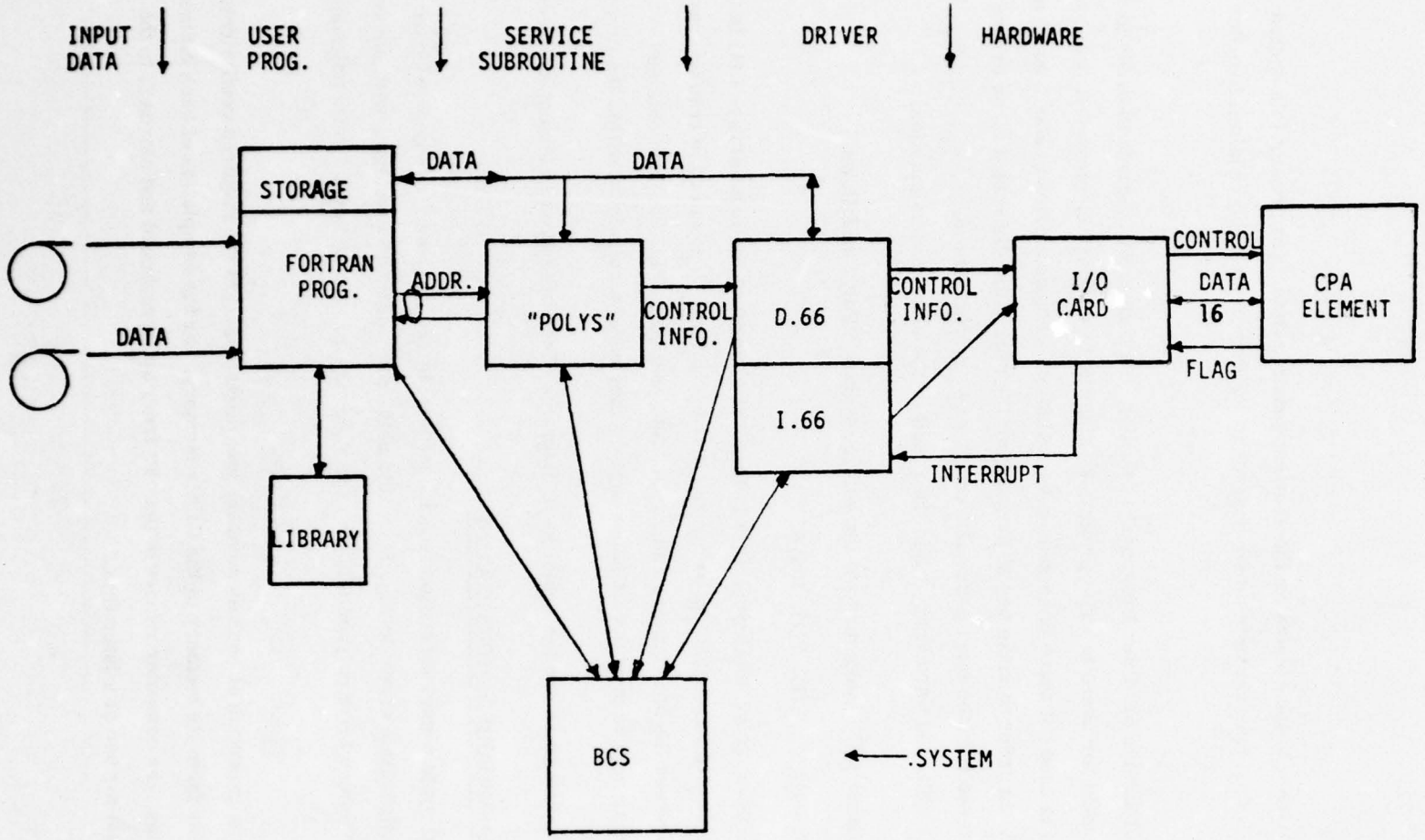


Fig. 25. Software System

transfer them to the element in the shortest possible time. (At present this system does not use DMA: however only minor changes to the driver would be required in order to do so).

The final module or "User Program" is written in a high level language in order to make it convenient for users to write programs using the brassboard element. At present Fortran is used but there is no reason why Algol or advanced forms of Basic could not be used. In order to make use of the element the user must dimension three arrays. Two of these hold the input points (X_1 and X_2) and coefficients (e.g., W_0, W_1, W_2, W_3, W_4, W_5) while the third array holds the result of the element's calculation.

A subroutine call is then made to the service routine "Poly" as follows:

Call Poly (N, D(1), C(1), R(1))

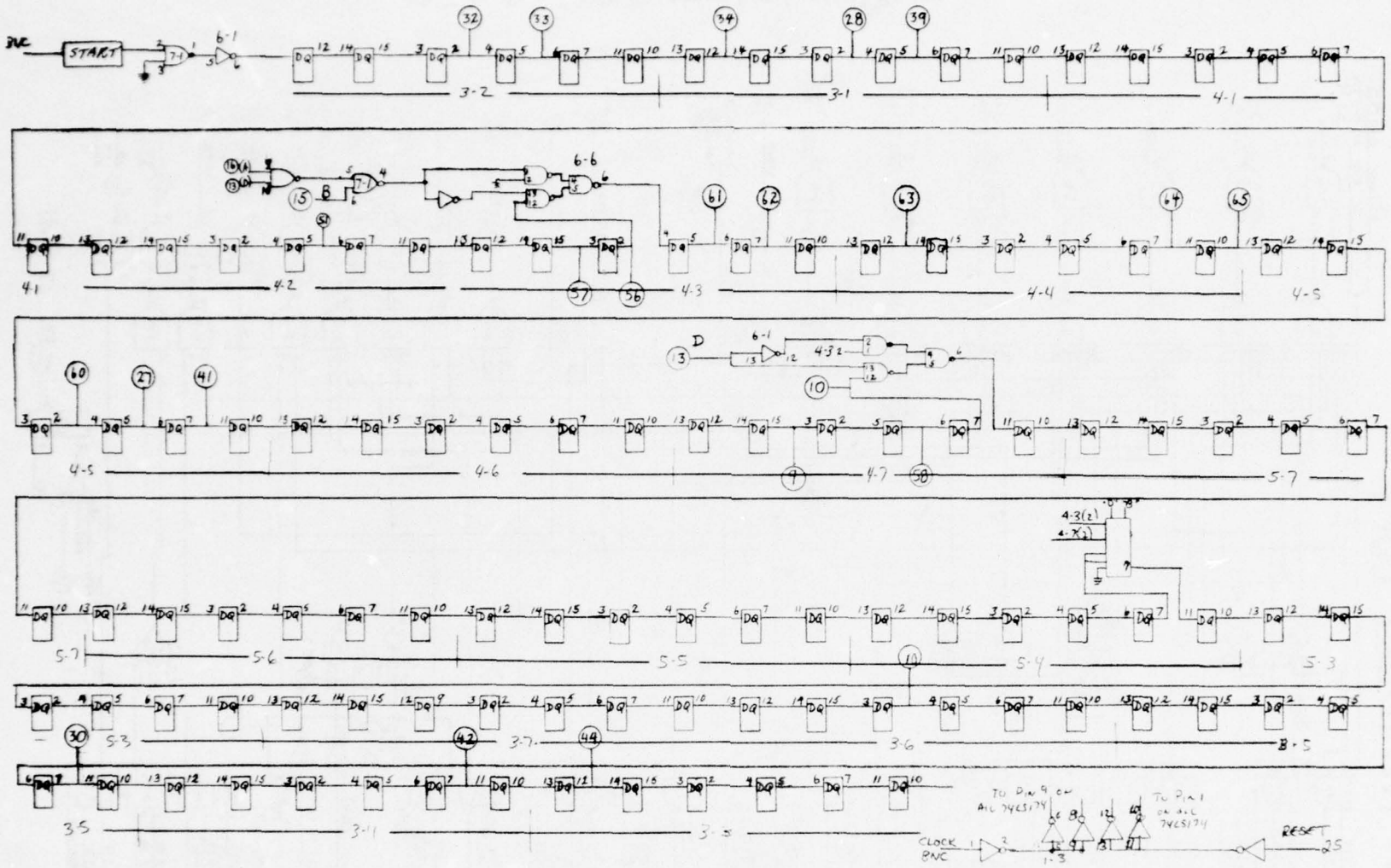
where N is the polynomial type, D(1) is the first element of the input array, C(1) is the first element of the coefficient array and R(1) is the first element of the result array, (This is where the answers are returned). As depicted in Fig. 25, both data and coefficients may be provided on paper tape or they may be calculated within the program.

Figs. 26 through 35 are the actual logic diagrams of the brassboard element as described.

5.6 BRASSBOARD DEMONSTRATION

A typical system implementation is demonstrated by the program "Array 5" with the actual CPA configuration shown in Fig. 36. This network involves 15 elements, with elements 1 thru 9 being 6 term polynomials (P1) and elements 10 thru 15 being 4 term polynomials (P2).

Before the execution of each element the computer transfers the required coefficients and data points from its memory to the CPA's memory. At the completion of each element's calculation, the computer retrieves the "Y" from the brassboard and stores it in the proper location(s) for use at a later time.



Fig, 26. Logic Diagram For Timing Board

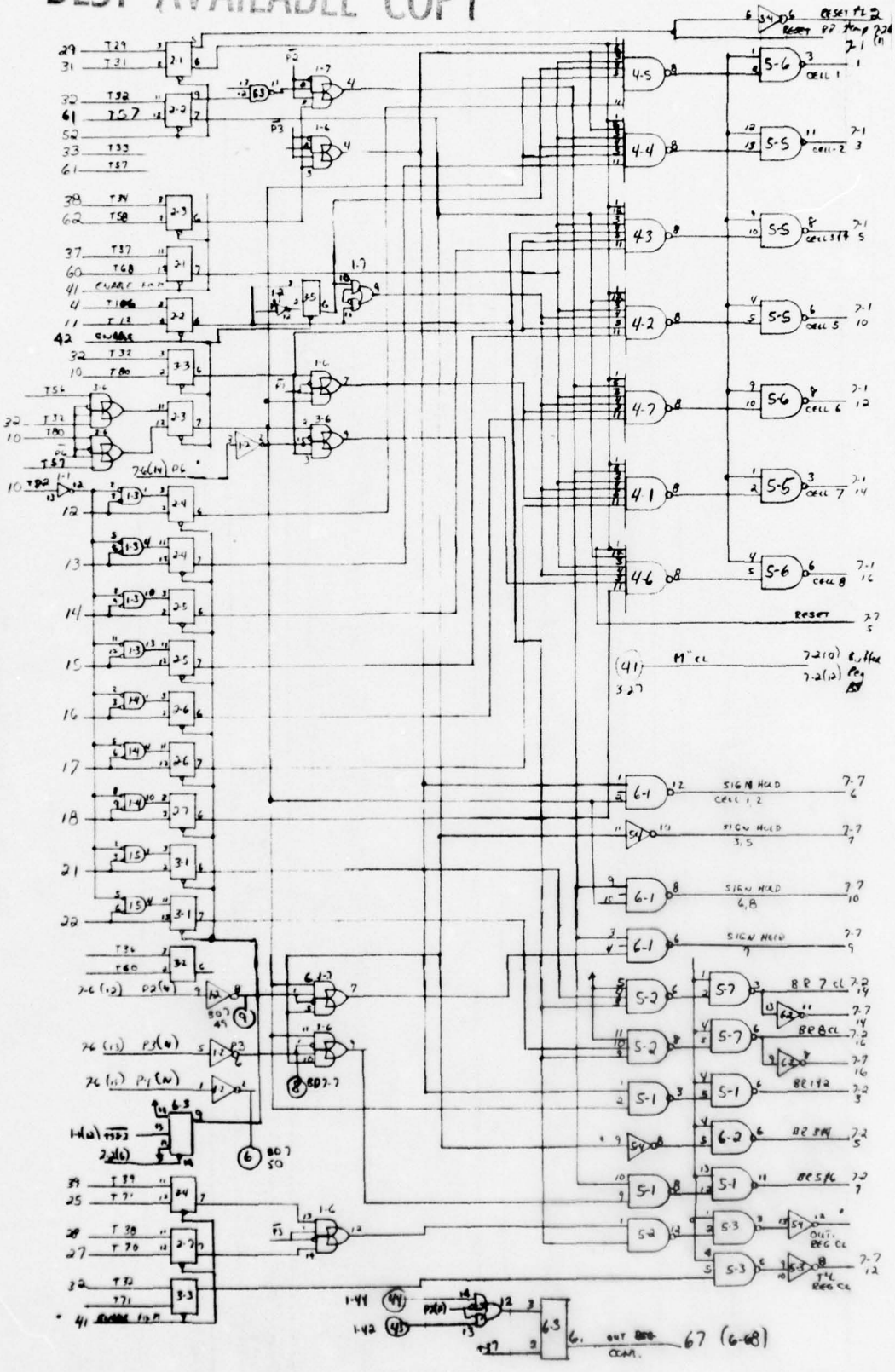


Fig. 27. Logic Diagram For Control Board

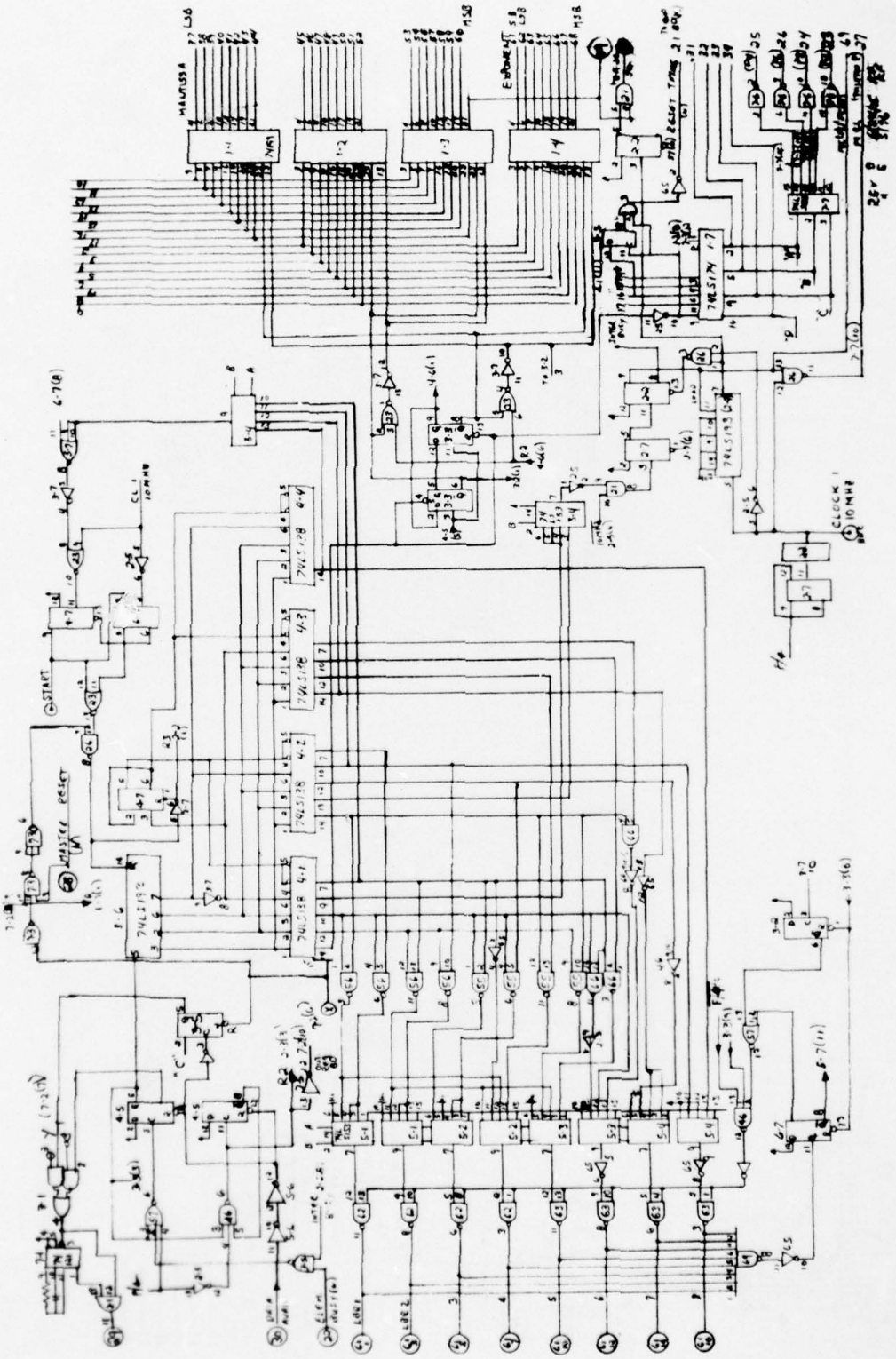


Fig. 28. Logic Diagram For Computer Interface

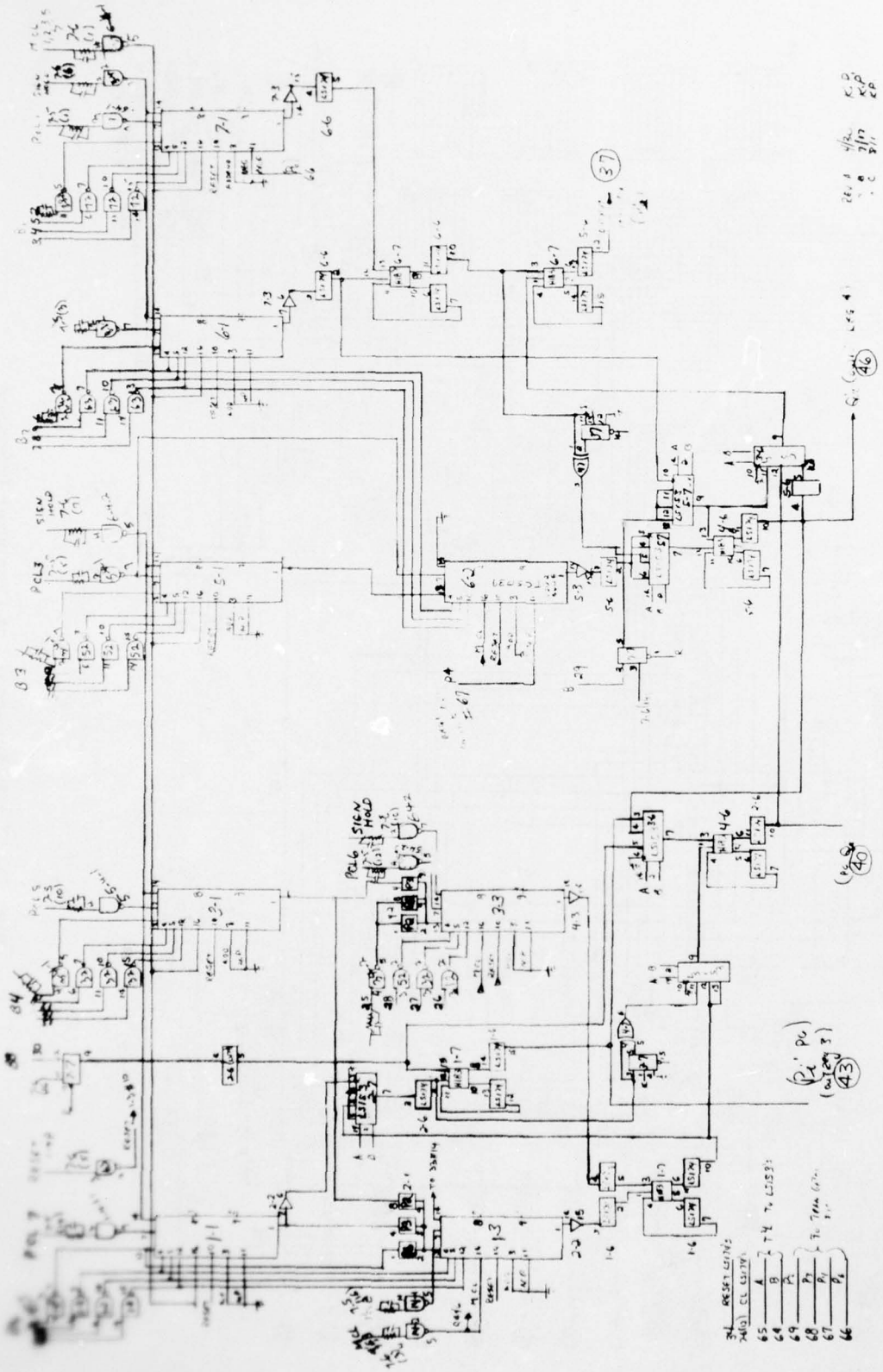


Fig. 29. Logic Diagram For Mantissa Processor

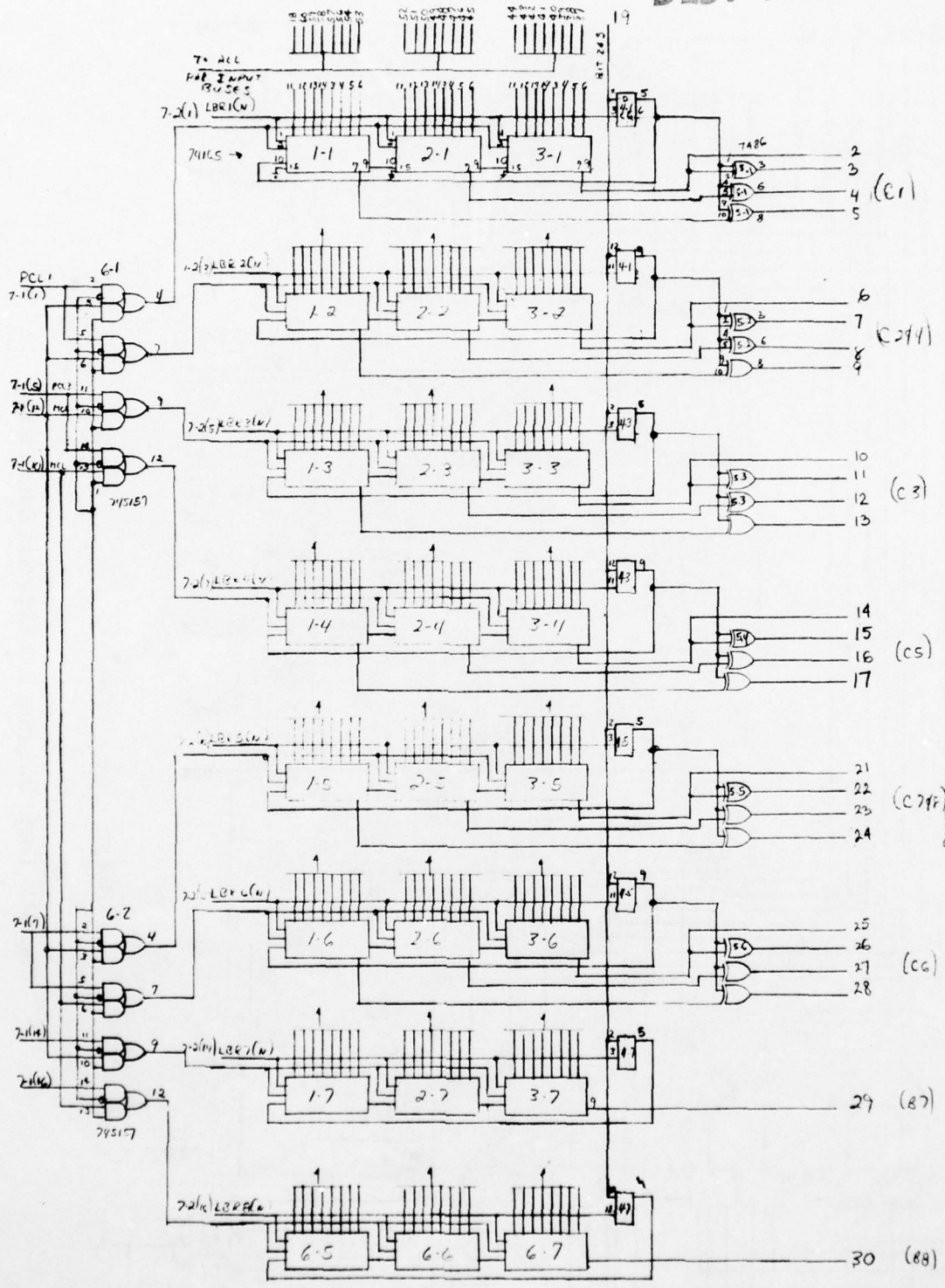


Fig. 30. Logic Diagram For Buffer Register Board

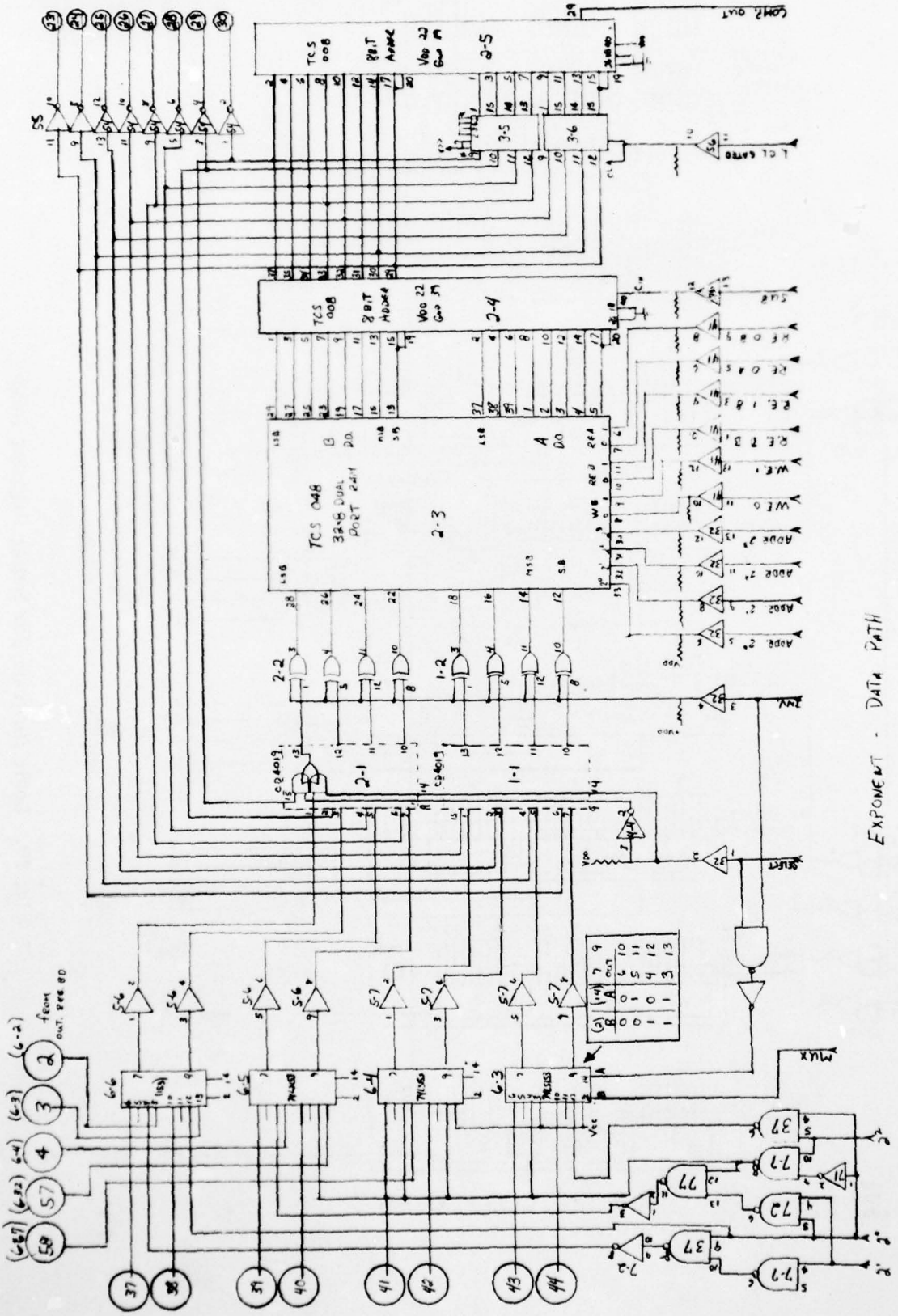


Fig. 31. Logic Diagram For Exponent Data Path

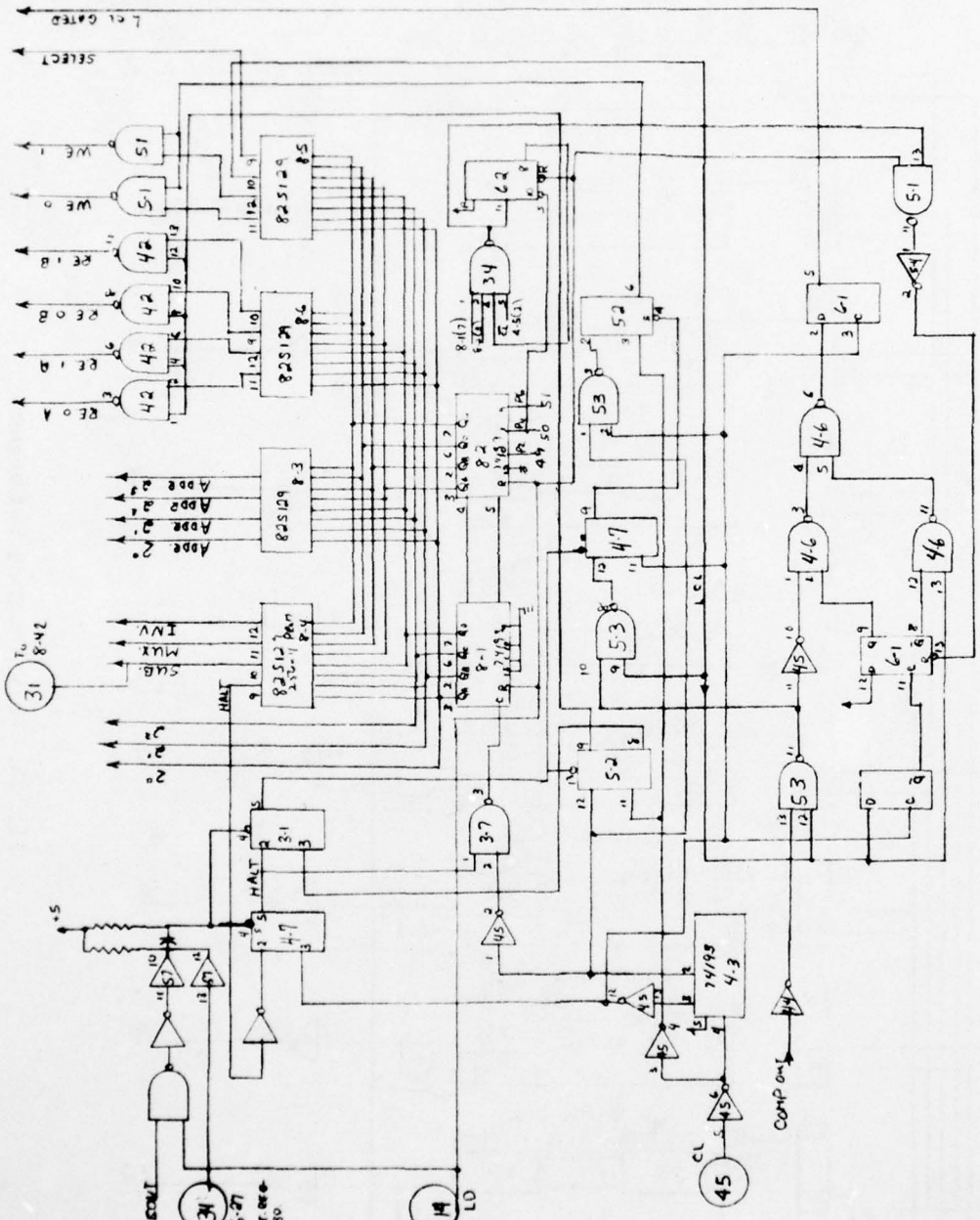


Fig. 32. Logic Diagram For Exponent Control Logic

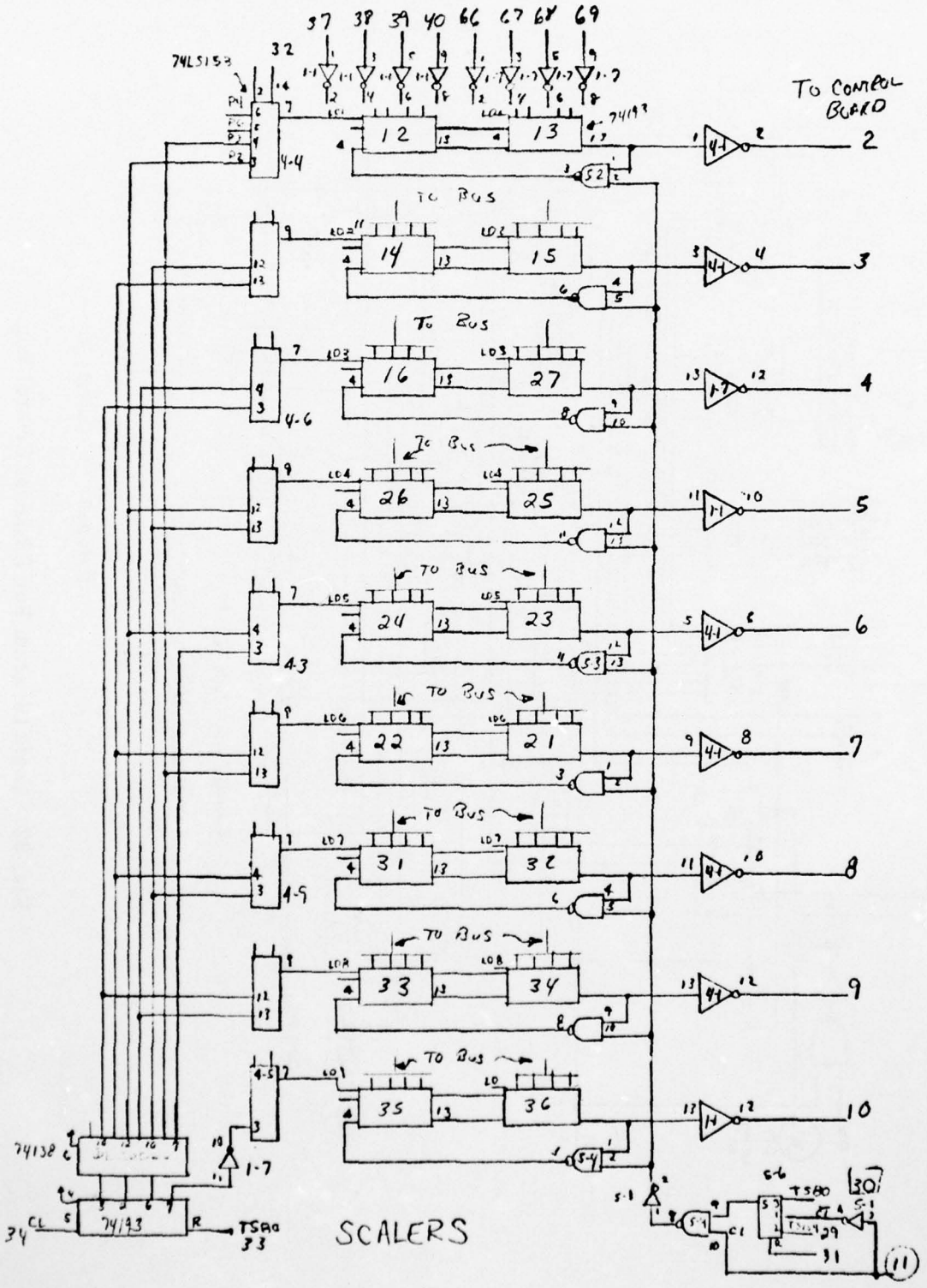


Fig. 33. Logic Diagram For Scalers

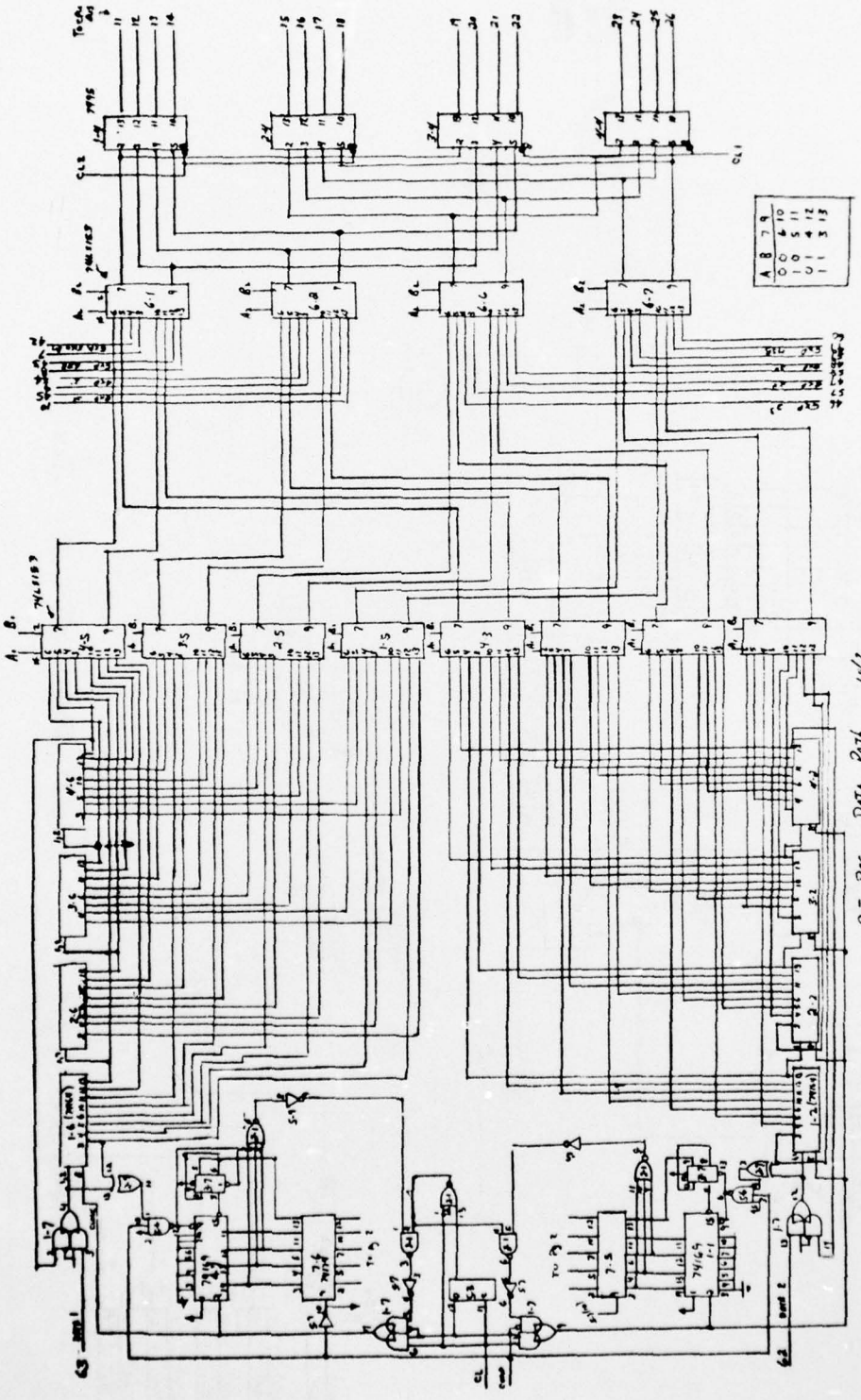


Fig. 34. Logic Diagram For Out. Reg. Data Path

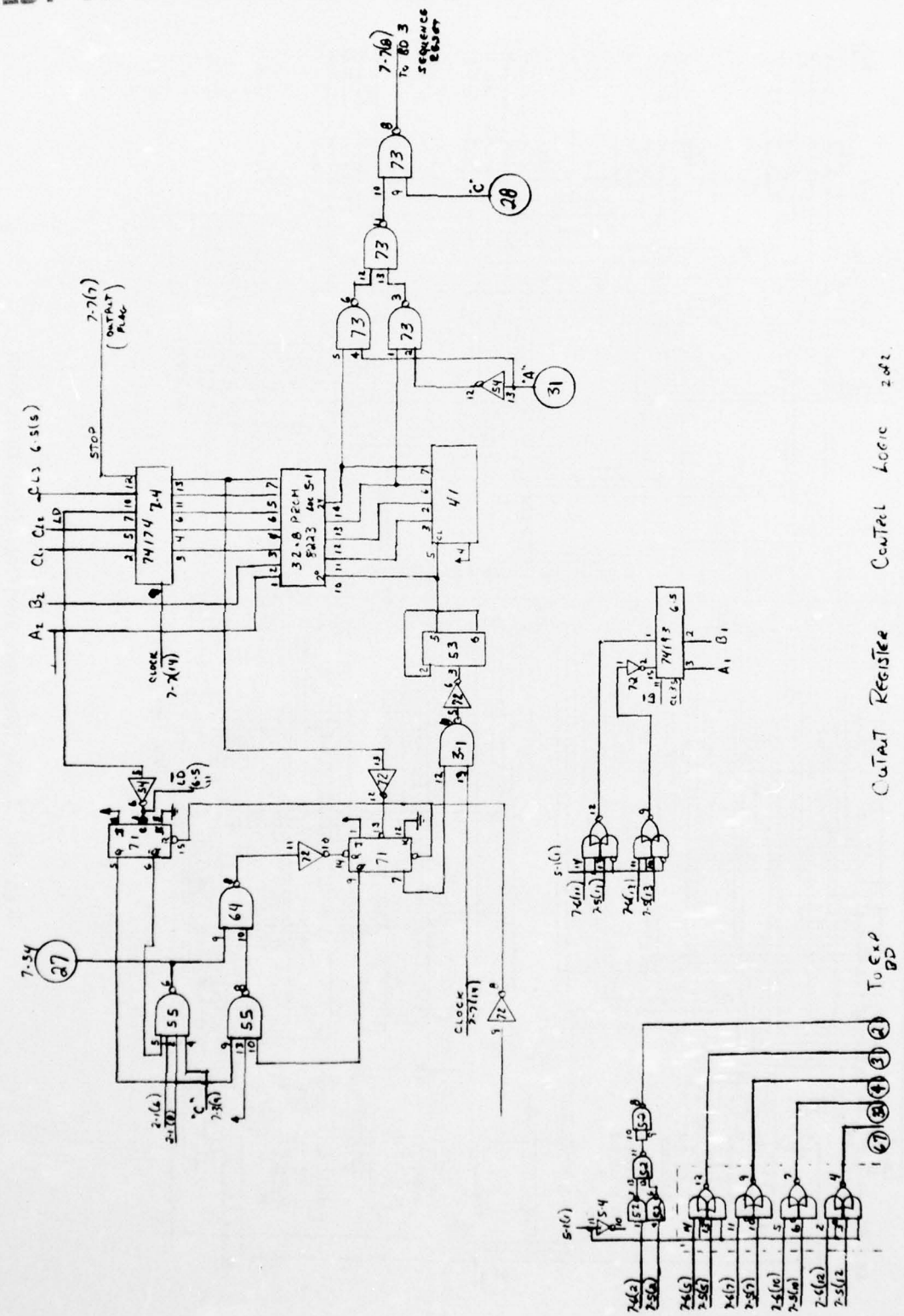


Fig. 35. Logic Diagram For Output Register Control Logic

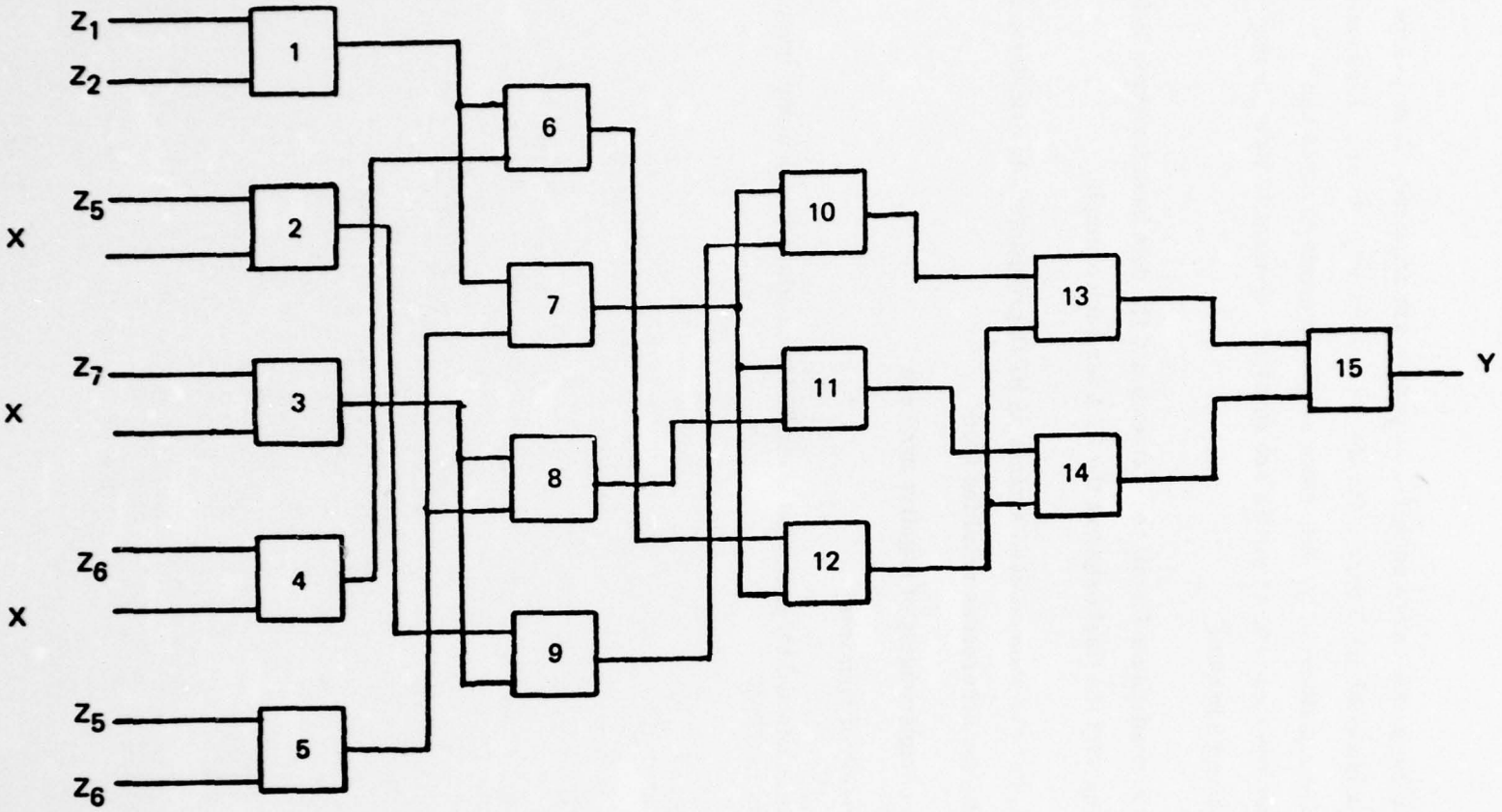


Fig. 36. Actual CPA Example

In order to implement the entire network, 13 passes are required. Nine passes for each of the 6 term polynomial and 4 passes for the 6 - 4 term polynomials. Elements 10 and 11 are implemented simultaneously, but, since before elements 13 or 14 can be implemented the outputs from elements 10, 11 and 12 are required, element 12 must be implemented along with a dummy element.

Table 8 lists the coefficients for all the elements and the data base for three different examples, along with the final output of the CPA for each example.

The final answers have been checked against an HP21 programmable calculator and are within 0.1% with the differences resulting from

1) one's complementing of negative numbers

2) truncation of mantissa

3) different internal bit precision (over 30 bit mantissa in calculator compared to 24 bits in CPA).

TABLE 8. CPA EXAMPLE COMPUTER CONTROLLED BRASSBOARD ELEMENT

```
PUT WEIGHT TAPE IN READER AND PRESS RUN
PAUSE

ELEMENT  W0        W1        W2        W3        W4          W5
  1   .91078,-.81804,-.40225,.37297,-.17185E1,-.12443E1/
  2   .52849,.24767E1,.14056,-.19501,-.32531E1,-.69229E-1/
  3   .17612E1,.17056E1,-.48766,-.38102,-.41839E1,.4748E-2/
  4   .15404E1,.22347E1,-.4337,-.46037,.19613,-.61202E-2/
  5   .3126E-1,.27059E1,-.33509,-.91401,-.34765E1,-.33589E1/
  6   -.35357,.81279,.59339,.33504,.25031,.14502/
  7   -.17459,.91152,.72919,.36352,.1779,.11306/
  8   .12494E-1,.31217,.53128,.44785,-.10771,-.62295E-1/
  9   -.13814,.36309,.25743,.97913,-.37447E-1,-.57359/
 10   -.76697E-2,.61635,.49344,.16947E-1/
 11   .4866E-1,.6997,.41215,-.7952E-1/
 12   .11885,.85086,.16804,-.16413/
 13   -.20998E-1,.27321,.7337,.26733E-1/
 14   -.46180E-1,.12274,.39234,.53003E-1/
 15   .51854E-2,.13361E1,-.33701,-.61725E-2/


PUT DATA TAPE IN READER AND PRESS RUN
PAUSE


.5085768      -.105837ZE1    -.2535387     .7852853E-1    .3965544E-1/
-.2961533     -.3075542      .3025527      .8101722E-1    .3002694E1

RESULT =  -.378995E+00
PUT DATA TAPE IN READER AND PRESS RUN
PAUSE


.5764495      -.9162722      -.1952316     .1114777       .6633897E-2/
-.1417897     -.2632610      .1530646      .1280857E-1    .33080E1
RESULT =  -.835903E+00
PUT DATA TAPE IN READER AND PRESS RUN
PAUSE


.6996189      .8986786       -.2145323     -.2635611      -.3561959E-1/
.1384989E-1   -.3285279      .2864978E-1   .6727891E-1    .522083E1
RESULT =  -.114112E+01
PUT DATA TAPE IN READER AND PRESS RUN
PAUSE
```

SECTION VI

TRADEOFF ANALYSIS

The tradeoffs which were conducted considered three basic architecture types, (Sec. V) using a variety of developmental or available parts. These three types include:

1) use of the serial-parallel multipliers operating in word parallel, bit serial form. This is the type embodied by the brassboard element. For this type, two different multiplier chips were considered; the RCA, CMOS/SOS circuit, (TCS 039, 24 bit) developed at the beginning of this project and a circuit that became available towards the end of this project, the bipolar, schottky T^2L (AMD 8-bit) part. Besides difference in bit length (and a considerably greater difference in power dissipation) these devices require somewhat different interface logic within the mantissa processor. They are characterized in the tradeoffs mainly by their difference in clock rate in curves that follow, with the TCS-039 considered to operate at 10 MHz and the bipolar unit considered to operate at 20 MHz. *

2) use of a parallel array multiplier. The element performance using these type multipliers was estimated for two state-of-the-art circuits also; the RCA (developmental) CMOS/SOS, 8X8 chip and the TRW (MPY-16) bipolar EFL, 16X16 chip. In this case, one complete parallel multiplier, used in a bit parallel, word serial fashion is used as a basis for the tradeoffs. That is 9, 8X8 chips for a 24-bit element, etc. In the case of the 16x16 chip, the performance was estimated only for the 16-bit precision fixed point element.

   Further assumptions for the all-parallel multiplier organizations in the tradeoffs are:

   using an 8X8 SOS chip

speed for 8X8 multiply	100ns
speed for 16X16 multiply	400ns
speed for 24X24 multiply	900 ns

   using the 16X16 bipolar chip

speed for 16X16 multiply	200ns

   Also, the use of the all-parallel 8X8 SOS chip assumes the use of the compatible family of developmental SOS devices for control, addition, and register storage.

*Each of these multiplier types will operate at somewhat higher speeds (15 MHz and 30 MHz respectively), but worst case and timing logic considerations dictated the more conservative estimates.

3) use of a microprocessor based element in conjunction with a parallel multiplier. This differs from case 2) in that a high speed micro (instruction time of about 300ns with SOS or bipolar) replaces all the element control logic. A low speed microprocessor based element (instruction time of about 2 to 3 μs) is later shown in relation to the other approaches for specific word length (16-bit).

Estimates of complexity (a measure of the total number of IC packages - custom LSI, or otherwise), speed, power dissipation and total chip cost were made with respect to bit length. Both fixed and floating point were considered for some of the cases. The complexity factor does not take into account element intraconnection, element interconnection or overall size. These consideration will of course affect overall system cost and reliability and gives the serial type multiplier architectural approach added advantage.

Fig. 37 shows the relative complexity for different precisions, for both fixed and floating point for the cases of the two types of serial and the 8X8 parallel multiplier. As noted, the floating point exponent is 4 bits for 8 bit mantissa precision and 8 bits for all other precisions. From a parts count point-of-view, the CMOS/SOS 24 bit unit is generally the most efficient, (except for 8 bits where a single AMD circuit for each multiplier suffices). The penalty for floating point can be seen in each case. (The present brassboard is the 32-bit floating point element using the 10 MHz multiplier.)

Fig. 38 gives the speed profiles of the alternate approaches. The speed penalty for floating point in the all-parallel multiplier approach is minor because parallel shift for binary point justification is assumed. More than 1 parallel multiplier can be employed to effectively double the speed especially for 24 or more mantissa bits. For example, whereas 9, 8X8 chips result in an 8 μsec execution period, 18 of them working in 2 independent multipliers can be used. Complexity and cost would however significantly increase.

The following performance estimates consider only fixed point elements and show the performance of the alternate all-parallel bipolar multiplier as well as the micro-processor based architecture in relation to the other approaches, and also go on to consider power and cost. One other important distinction is also made. The succeeding curves consider an element with sufficient memory so that it can be multiplexed to form a net with up to 20 elements (the multiplexing aspect as discussed earlier in this report.)

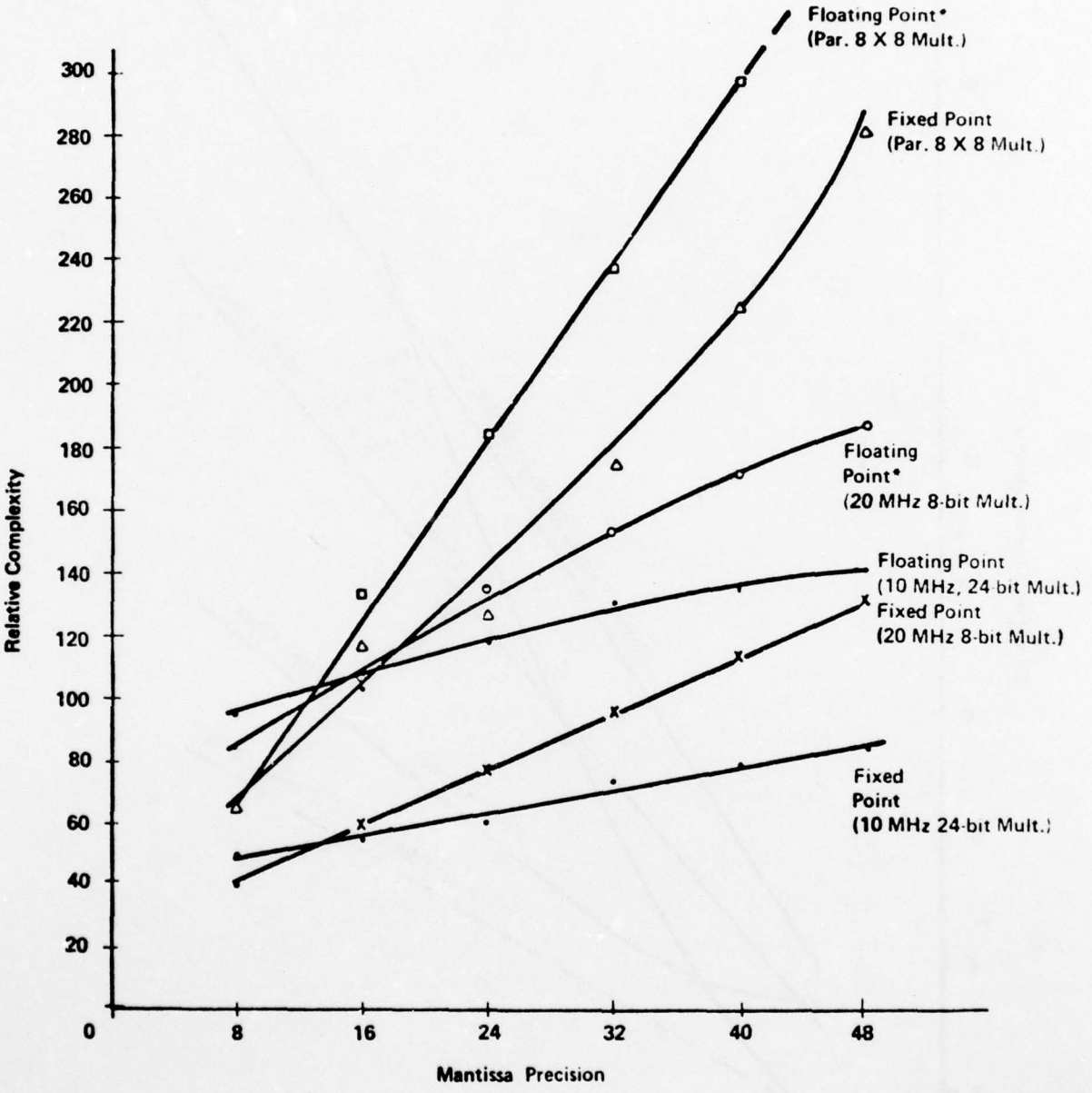


Fig. 37. CPA Complexity for Various Precisions

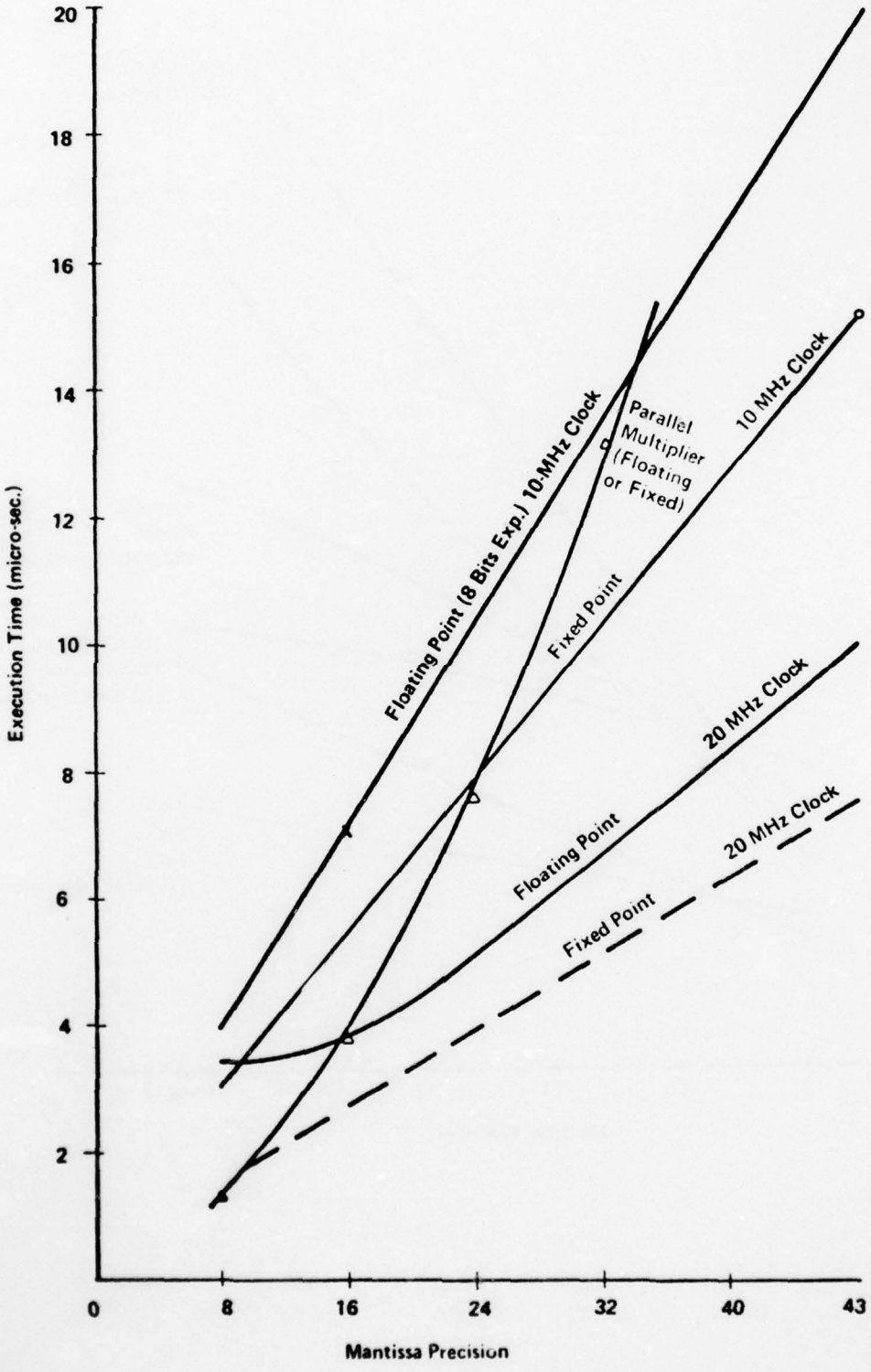


Fig. 38. CPA Element Speed

AD-A042 256 RCA GOVERNMENT COMMUNICATIONS SYSTEMS SOMERVILLE N J --ETC F/G 9/2
LSI ELECTRONICALLY PROGRAMMABLE ARRAYS: CONFIGURABLE POLYNOMIAL--ETC(U)
JUN 77 D HAMPEL, K PROST, R L BARRON F33615-73-C-1089
UNCLASSIFIED AFAL-TR-76-228 NL

2 OF 4
AD A042256

The features of each system estimated now are summarized below:

1) Serial-Parallel 24-bit SOS/CMOS multiplier (10 MHz)

memory - high speed RAMS for serial inputs (capable of implementing a minimum of 20 elements)

mantissa - 8 serial/parallel multipliers with multiplexers to reconfigure the network to implement the 6 term or 4 term polynomial or a FFT.

output registers - uses tri-state T^2L registers.

2) Serial-Parallel 8 bit T^2L multiplier (20 MHz)

Similar to approach #1 with the exception of the multipliers used.

These architectures are based on our present breadboard model.

3) Parallel multipliers configuration using the 8X8 CMOS/SOS multipliers

This approach is based on custom LSI devices developed within RCA and include the following:

8X8 parallel multipliers (expandable)	TCS077
18 stage parallel in-parallel out registers	TCS015
8 stage parallel adders	TCS000
1029 X 1 RAM (commercially available)	MWS5501D
256 X 4 RAM (commercially available)	MWS4440D
multiplexers (commercially available)	CDS4066

4) Parallel system using TRW's 16 X 16 multiplier (MPY16A) Similar to the SOS multiplier system but incorporates T^2L parts beside the EFL multiplier.

5) Micro-processor based system

This approach is based on a high speed micro-processor as discussed in conjunction with separate hardware multipliers, adders, etc. for use in signal processing. Typical of this approach is the new RCA ATMAC, or other bipolar units soon to be available.

The complexity estimates for these assumptions (Fig. 39) include all necessary control logic for stand-alone elements. The previous two curves were based essentially on fully populated CPA's where common control was possible. It is seen that the micro-processor scheme is advantageous from a total parts point of view, while all other alternatives are grouped together for 16 bit precisions. A somewhat reverse trend is noted for speed (Fig. 40).

The cost for the various T^2L parts are based on published price lists while the cost of the custom SOS devices have been estimated from between \$100 - \$150 per unit in small to moderate quantities (up to 100). The component costs depend to a large extent as to production volume. For low to moderate volumes, catalog IC's will always be substantially less than any customized part such as the SOS or EFL multipliers or high performance developmental micro-processors. Fig. 41 reflects this trend and shows that all alternatives are close in cost for 16-bit precision for reasons of component count or custom circuit usage. The use of the TRW multiplier (a single point or all curves for 16 bits) would result in a lower cost element because it can be used with commercially available TTL. Finally, the power trends, Fig. 6-6, show the obvious advantage of CMOS/SOS based architectures.

In summary, these curves (Fig. 39 to 42) are based on multiplexed or fully populated CPA architectures (as opposed to the curves of Figs. 37 and 38). The micro-processor based element depends largely on the technology used. The power curves would be different for a bipolar device, while speed, complexity, and cost might not vary that much. Production volume must be more closely tied in to specific estimates based on developmental LSI components.

To consolidate this data and put it in perspective with a low cost, low speed microprocessor based element, Table 9 was prepared. The all-parallel multiplier organization is not included here since it was generally out-performed by the serial approach. Here, we consider average complexity and speed, and relative cost. Only the low cost, low speed micro approach does not have a necessary chip assumed for its performance. This would be a serial-parallel multiplier (similar to those used in approach 1, but customized for interfacing with the micro). While the defined figure of merit (the lower, the better) indicates

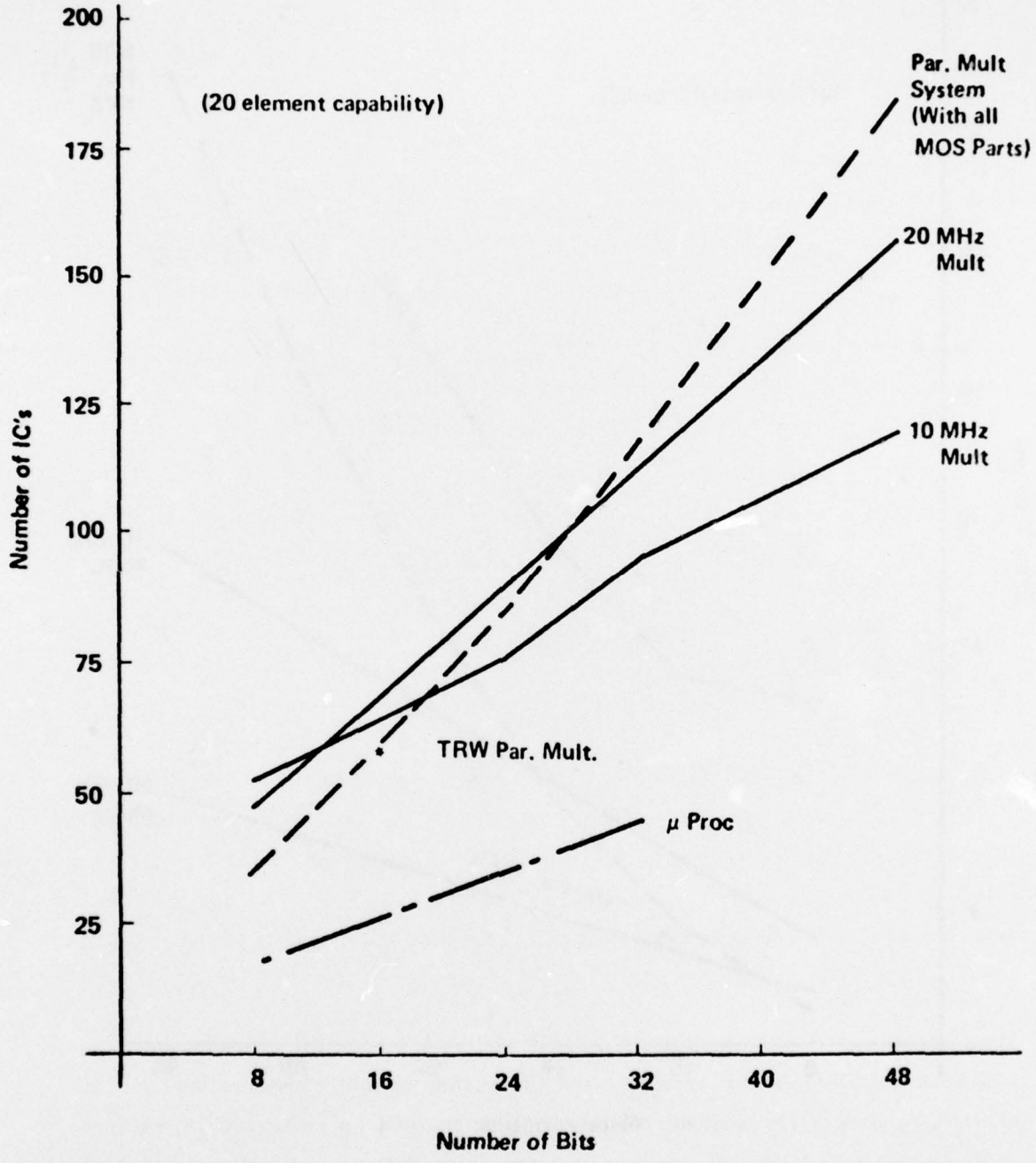


Fig. 39. Complexity vs. Bit Length

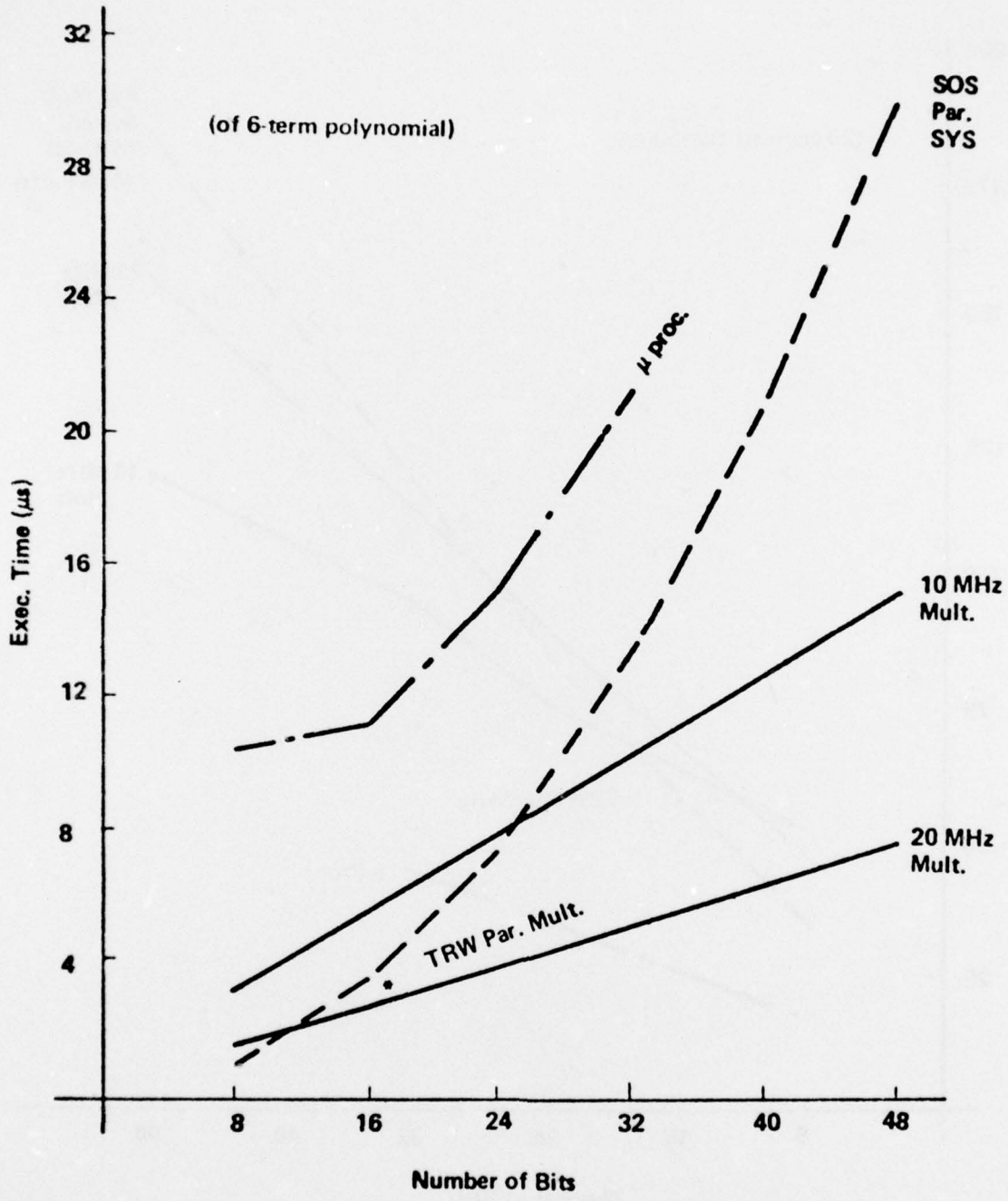


Fig. 40. Speed vs. Bit Length (of 6-Term Polynomial)

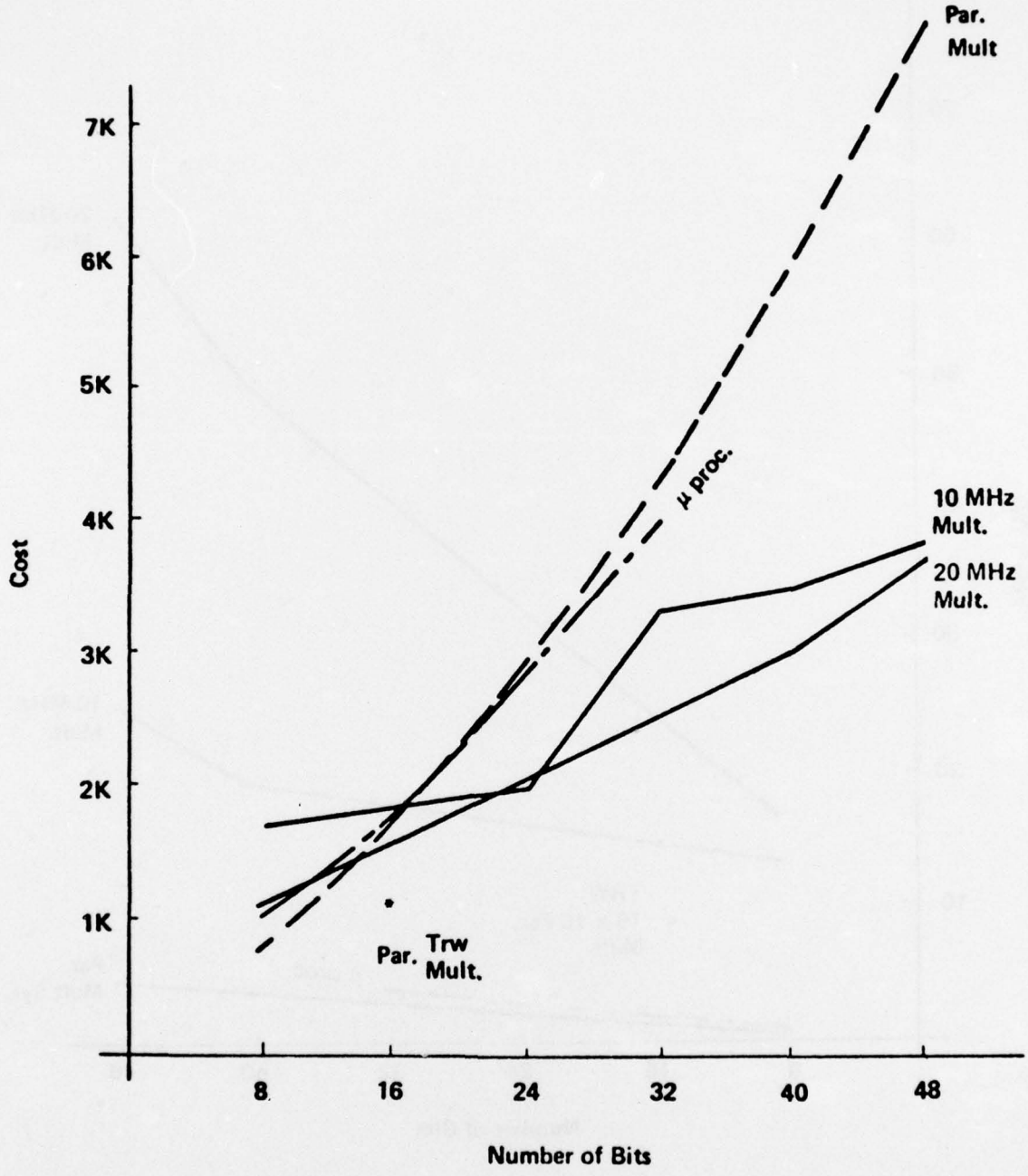


Fig. 41. Component Cost vs. Bit Length

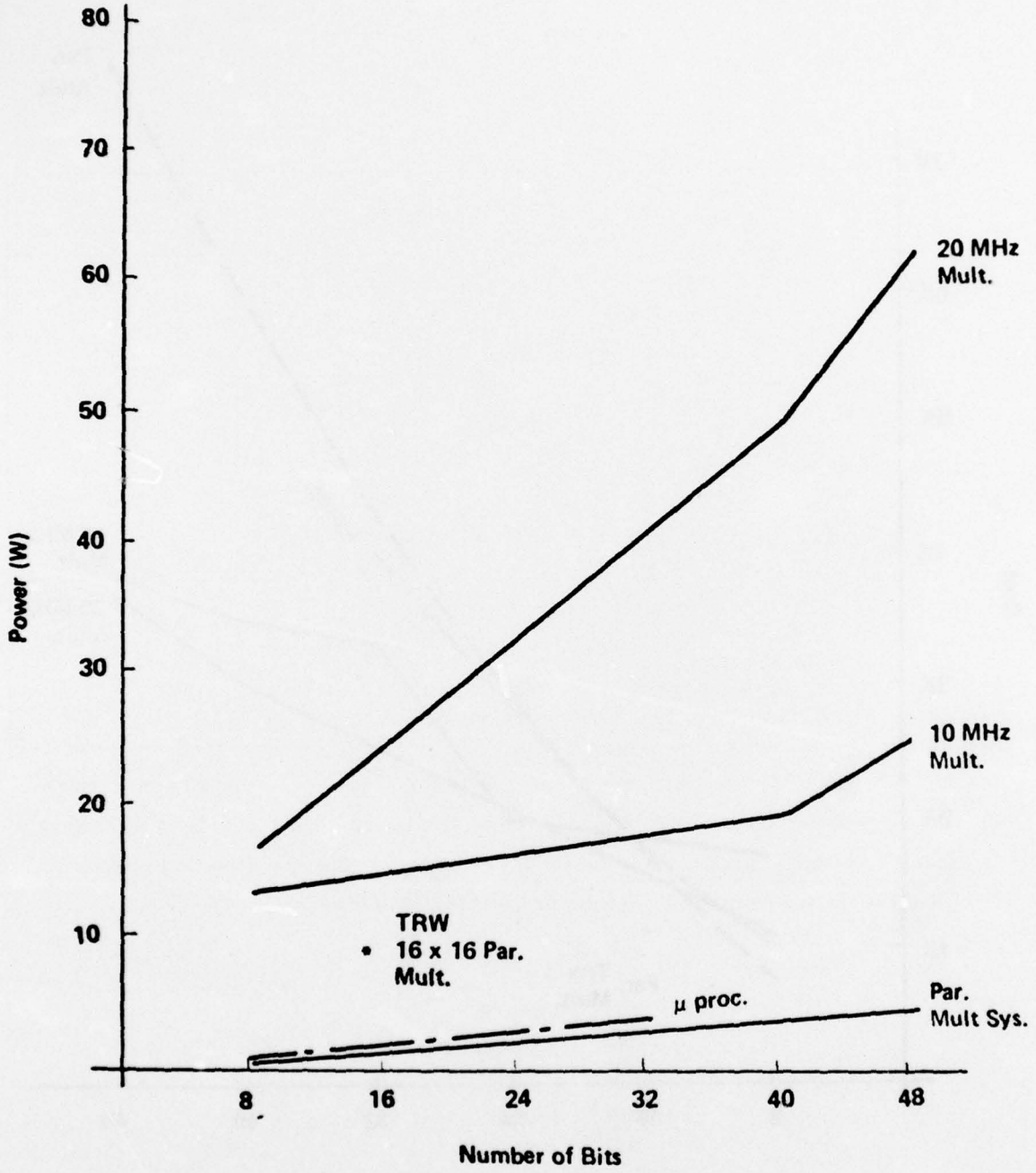


Fig. 42. Power vs. Bit Length

TABLE 9. PERFORMANCE OF CPA ELEMENT ARCHITECTURES

Architecture Type (for 16 Bit Fixed Point)	No. of IC's	Speed	Cost	Figure of Merit IC's Speed Cost
1. Serial Multiplier	33	3μs	X	99
2. High Speed Signal Proc. Micro. - (16 Bit Org., 300ns inst. time) (ATMAC or Equiv.)	12	10μs	X	120
3. Low Cost Micro-Proc. (8 Bit Org., 2-3μs inst. time) (1802 or Equiv.)	5	150μs	X/8	94
Elements for use in fully Populated Arrays 1. Uses all Available Parts 2. Developmental; Large Connectivity, All-Parallel Circuitry 3. Based on Availability of Compatible Serial Mult. (Under Dev.)				

the best efficiency (if power dissipation were included it would even show this much more dramatically) of the low-cost micro, the serial multiplier approach, in today's technology, is about the same in overall performance. It is more likely that any specific implementation would be chosen on the basis of speed requirement, of course.

SECTION VII

ANALYSIS OF FIXED-POINT PROCESSING OF NESTED POLYNOMIALS

7.1 INTRODUCTION

Nested polynomials have been utilized to create nonlinear functions which "model" the underlying relationships implicit in a "data base" or matrix of numerical observations [Ref 1]. The data base can be thought of as consisting of M rows, where each row is an "observation" of the n + 1 variables y, x_1, x_2, ..., x_n. Algorithms have been established to "train" or create a nested polynomial function

$$\bar{y} = p(x_1, x_2, \ldots, x_n) \tag{7.2}$$

such that $\bar{y} \stackrel{\sim}{=} y$ for each of the M observations in the data base (where "$\stackrel{\sim}{=}$" denotes some criterion of fitting, such as minimum square error), and $\bar{y} \stackrel{\sim}{=} y$ *is expected to hold for any* <u>new</u> observation (or observation which was not used by the training algorithm). This latter requirement is sometimes referred to as "overfit avoidance".

The nested polynomial, $\bar{y} = p(x_1, x_2, \ldots, x_n)$, is a composition of elementary polynomials or "elements". An element, for example, could be the six-term, two variable polynomial

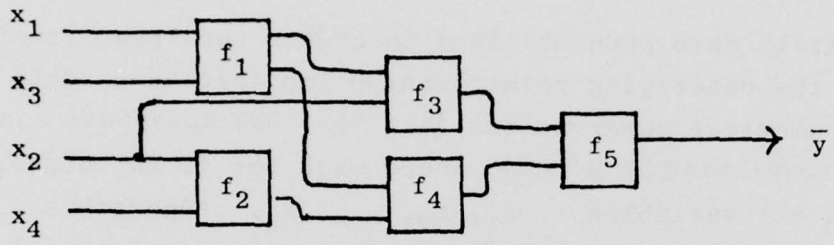
$$f_K(z_1, z_2) = w_{0,k} + w_{1,k}z_1 + w_{2,k}z_2 + w_{3,k}z_1z_2 + w_{4,k}z_1^2 + w_{5,k}z_2^2$$

The elements are used recursively, or "nested", to form the nested polynomial function.

For example, a nested polynomial function of four variables may be:

$$\bar{y} = P(x_1, x_2, x_3, x_4) = f_5\left\{f_3[f_1(x_1, x_3), x_2], f_4[f_1(x_1, x_3), f_2(x_2, x_4)]\right\}$$

This can also be expressed by a diagram:



It is, of course, possible to expand this function, that is, expand it directly as a polynomial in the four variables x_1, x_2, x_3, x_4. Such an expansion will consist of a great many terms, and is computationally inefficient compared to the nested element representation.

Recently, LSI hardware has been developed to compute nested polynomials of a certain type [Refs. x,x]. This hardware uses 32-bit floating point arithmetic but the design could be modified to perform fixed-point computations. Fixed-point hardware is generally faster, less expensive, and requires less power to operate than floating point hardware.

7.2 OVERFLOW AND ACCUMULATION OF ROUND-OFF ERROR

There are two potential hazards of fixed-point computation which must be considered: overflow, or the computing of a number, x, outside of the range $|x| \leq 1$; and the accumulation of round-off error beyond an acceptable tolerance.[1/]

In Part 1, it will be shown that by observing certain restrictions while "training" a nested polynomial to model a given data base, the resulting nested polynomial is guaranteed never to overflow. These restrictions are shown to be very mild in that the modeling

1/ Some fixed point processors omit +1 from the permissible range.

capability of the training algorithm is not affected for all practical purposes.

In Part 2, the accumulation of round-off error will be examined, and in Part 3, the word-length requirement of a fixed-point nested polynomial processor will be stated in terms of (a) the accuracy of the data, (b) the number of "layers" of the nested polynomial to be implemented, and (c) the restrictions imposed on the training algorithms. Nested polynomials consisting entirely of six-term elements (as above) will be considered. Nested polynomials composed of one or more different types of elements can be analyzed along similar lines.

Part 1 - Restrictions on Model Training Which Prevent Overflow

Triangle inequality -- For any two numbers x and y, it is always true that

$$|x+y| \leq |x| + |y|$$

Let $x = a + b + c$ and $y = -c$. Then:

$$|a + b + c + (-c)| \leq |a + b + c| + |-c|$$

$$|a + b| \leq |a + b + c| + |c|$$

$$|a + b| - |c| \leq |a + b + c|.$$

This form of the triangle inequality will be used in what follows.

Theorem 1:

Let $f(x) = ax^2 + bx + c$, and $|x| \leq 1$, $|f(x)| \leq A$. Then:

$$|a| \leq 2A, \; |b| \leq 2A, \; |c| \leq A.$$

Proof:

(1) $f(0) = c$, $|f(x)| \leq A \rightarrow |c| \leq A$.

(2) $f(1) = a + b + c$, $|f(x)| \leq A \rightarrow |a + b + c| \leq A$.

$|a + b| - |c| \leq |a + b + c| \leq A$.
$|a + b| \leq A + |c| \leq 2A$
$|a + b| \leq 2A$.

(3) $f(-1) = a - b + c \rightarrow |a - b| \leq 2A$ as in (2) above.

(4) $|(a + b) + (a - b)| \leq |a + b| + |a - b| \leq 4A$

$\rightarrow 2|a| \leq 4A$

$|a| \leq 2A$.

(5) $|(a + b) - (a - b)| \leq |a + b| + |-(a - b)| \leq 4A$.

$\rightarrow |b| \leq 2A$

Theorem 2:

Let

$$f(x_1, x_2) = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2$$

and $|x_1| \leq 1$, $|x_2| \leq 1$, $|f(x_1, x_2)| \leq A$.

Then:

$|w_0| \leq A$, $|w_1| \leq 2A$, $|w_2| \leq 2A$, $|w_3| \leq A$,
$|w_4| \leq 2A$, $|w_5| \leq 2A$.

Proof:

(1) $|f(0,0)| = |w_0| \leq A.$

(2) $f(x_1,0) = w_0 + w_1x_1 + w_4x_1^2$; $|f(x_1,0)| \leq A.$

Therefore from Theorem 1, $|w_1| \leq 2A$, $|w_4| \leq 2A.$

(3) $f(0, x_2) = w_0 + w_2x_2 + w_5x_2^2$; $|f(0, x_2)| \leq A.$

Therefore from Theorem 1, $|w_2| \leq 2A$, $|w_5| \leq 2A.$

(4) Finally, to bound w_3, consider

$$[f(1, 1) - f(-1, -1)] - [f(1, -1) + f(-1, 1)] = 4w_3$$

Therefore:

$$|4w_3| \leq |f(1, 1)| + |f(-1, -1)| + |f(1, -1)| + |f(-1, 1)|$$

$$\leq 4A$$

$$|w_3| \leq A.$$

Theorem 3:

Let

$$f(x_1, x_2) = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2$$

and

$$|x_1| \leq 1, |x_2| \leq 1, |w_0| \leq A, |w_1| \leq 2A, |w_2| \leq 2A,$$

$$|w_3| \leq A, |w_4| \leq 2A, |w_5| \leq 2A.$$

Then:

(a) $|f(x_1, x_2)| \leq 10A$

(b) In computing $f(x_1, x_2)$, no number greater in absolute value than 10A can be formed.

Proof:

$$|f(x_1, x_2)| \leq |w_0| + |w_1x_1| + |w_2x_2| + |w_3x_1x_2| + |w_4x_1^2| + |w_5x_2^2|$$

$$\leq A + 2A + 2A + A + 2A + 2A$$

Therefore $|f(x_1, x_2)| \leq 10A$, and each term in the sum is bounded by A or 2A.

Restrictions on Training a Nested Polynomial to Model a Data Base--

(a) Data Base Scaling: Each of the n + 1 variables in the data base must be scaled prior to generating the nested polynomial. The scale functions $S_1 \ldots S_n$ must be monotonic and must map the domain of each input variable into [-1, 1]. The scale function S_y must be monotonic and must map the domain of y into [-B, B], where B < 1. The purpose of B is discussed in (d) below. Primes will be used to denote the scaled values:

$$x_1' = S_1(x_1)$$

$$x_2' = S_2(x_2)$$

$$\vdots$$

$$x_n' = S_n(x_n)$$

$$y' = S_y(y)$$

Note that the domain of each of the N + 1 variables must be estimated from the known occurrences of these variables (in the data base or elsewhere), or from known physical limitations.

Once these scale functions are known, the desired function $y = p(x_1, x_2, \ldots, x_n)$ may be realized by training a nested polynomial, q, on the scaled data. That is, if $y' = q(x_1', x_2', \ldots, x_n')$, then the desired function, p, is the result of the successive transformations:

$$x_1' = S_1(x_1)$$

$$\vdots$$

$$x_n' = S_n(x_n)$$

$$y' = q(x_1', x_2', \ldots, x_n')$$

$$y = S_y^{-1}(y').$$

In addition to the scaling of data base variables, it is necessary to scale the element outputs within the training algorithm and within the resulting nested polynomial. This intermediate scaling will be discussed in (e) below.

(b) Element Verification: Each element which may be used in the nested polynomial, q, must be inspected to insure that its coefficients satisfy all of the following:

$$|w_0| \leq 0.1$$

$$|w_1| \leq 0.2$$

$$|w_2| \leq 0.2$$

$$|w_3| \leq 0.1$$

$$|w_4| \leq 0.2$$

$$|w_5| \leq 0.2$$

An element which fails this test cannot be used. This element verification test is the only restriction that must be added to the training algorithm in order to guarantee fixed-point realization. (The scaling requirements of part (a) and (e) do not restrict the performance of the training algorithm.) The effect of the element verification test on the modeling capability of the training algorithm will be discussed below.

(c) Word Length Versus Training Algorithm Capability - A Tradeoff: Theorems 2 and 3 show the relationship of function boundedness and coefficient boundedness for the six-term element. This relationship in turn suggests a tradeoff between processor word length and training algorithm capability, as larger word lengths enable greater scaling (smaller values of A in Theorem 3), which in turn make the element verification test less restrictive.

(d) Single-Element Model: The purpose of scaling the y-data into a smaller range than the x-data can be appreciated by considering a single-element model. We will choose B = 0.0625. Assume that all variables in the data base span the range [-1,1], and that each number in the data base is exactly expressed in v bits. Then, in order to scale the y's such that $|y'| \leq 0.0625$, it is necessary to have a word length of at most 4 + v bits ($0.0625 = 1/16 = 2^{-4}$). Now if the element does not satisfy the verification test of (b), this implies (by Theorem 2 , with A = 0.1), that the element must somewhere assume a value greater than 0.1 in absolute value within its domain. Since the element is itself the "model" of $y' = q(x_1', x_2', \ldots, x_n')$, then such an excursion would imply that the rejected element was, at least in some region of its domain, more than 60 percent greater (in absolute value) than the desired model.

Similarly, if B = 1/32, then 5 + v bits will be required, and only elements which are more than 220 percent greater (in absolute value) than the desired model may be rejected.

For all practical purposes, B = 1/16 will not unduly restrict the training algorithm, and therefore 4 + v bits will be required to guarantee that overflow cannot occur.

(e) Nested Polynomial Model: In general, the desired function, q, is approximated by a nest of elements rather than by a single element. In this case, the "intermediate variables," or outputs of elements which are used as inputs to successive elements, must also be scaled. This is because the ratio of the range of element output to the range of element input must be the same for each element, and this ratio is given by B. In the training algorithm, each element output which is to be used as an input to successive elements must be scaled such that its range is nearly [-1, 1]. In the resulting model, the inverse scale functions are applied to these intermediate variables. By using scale factors which are powers of 2, the intermediate-variable scaling reduces to simple shift operations.

Part 2 - Analysis of Rounding Errors

The notation follows Wilkinson's Rounding Errors in Algebraic Processes (Prentiss-Hall, 1963). The equivalence sign ($\equiv$) is used in expressions of the form

$$S \equiv C(a_1 + \varepsilon_1, a_2 + \varepsilon_2, \ldots, a_n + \varepsilon_n)$$

where the numbers $a_1, a_2, \ldots, a_n$ are initial data or previously computed quantities, and S is the exact value of the computed quantity, C. We then try to find inequalities which bound the ε's. See Wilkinson's reference to "backwards analysis" for details.

Rounding Errors in Fixed-Point Arithmetic -- Assume an arbitrary word length of t bits. There is no rounding error in addition or subtraction, i.e.

$$\text{fix}(a + b) \equiv a + b$$

$$\text{fix}(a - b) \equiv a - b$$

In multiplication, the rounding error depends upon the rounding procedure used. We will assume that a 2t-bit product is formed, 2^{-t-1} is added, and the t most significant bits are retained. In this case,

$$\text{fix}\,(a \circ b) \equiv a \cdot b + \varepsilon;\ |\varepsilon| \leq 2^{-t-1}$$

Note that overflow cannot occur in multiplication, but can occur in addition or subtraction.

Rounding Errors in Computing a Six-Term Element -- where

$$|x_1| \leq 1,\ |x_2| \leq 1,\ |w_i| \leq 0.2;\ i = 1, 2, 4, 5;$$

$$|w_j| \leq 0.1;\ j = 0,3.$$

(1) Ideal case: If the processor has the capability of performing (t x 2t)-bit multiplication and accumulating 3t-bit products, then rounding takes place only at the very last step, when the 3t-bit result is rounded to t bits. In this case,

$$\text{fix}[f(x_1, x_2)] \equiv f(x_1, x_2) + \varepsilon;\ |\varepsilon| \leq 2^{-t-1}$$

(2) Near-ideal case: It is likely that a t-bit processor would not have the ideal-case capability. More reasonable capability would be (t x t)-bit multiplication and 2t-bit accumulation. For this case, let fix* denote the 2t-bit result of an operation. Then

$$\text{fix}[f(x_1, x_2)] \equiv \text{fix}^*[f(x_1, x_2)] + \varepsilon_1;\ |\varepsilon_1| \leq 2^{-t-1}$$

Note that:

(a) $\text{fix}^*(w_1 x_1) \equiv w_1 x_1$

(b) $\text{fix}^*(w_2 x_2) \equiv w_2 x_2$

(c) $fix^*(w_3x_1x_2) \equiv w_3(x_1x_2 + \varepsilon_3) = w_3x_1x_2 + w_3\varepsilon_3$

Let $E_3 = w_3\varepsilon_3$. Since $|w_3| \leq 0.1 < 2^{-3}$, $|\varepsilon_3| \leq 2^{-t-1}$,

then $fix^*(w_3x_1x_2) = w_3x_1x_2 + E_3; |E_3| \leq 2^{-t-4}$.

(d) $fix^*(w_4x_1^2) \equiv w_4(x_1^2 + \varepsilon_4) = w_4x_1^2 + w_4\varepsilon_4$.

Let $E_4 = w_4\varepsilon_4$. Since $|w_4| \leq 0.2 < 2^{-2}$, $|\varepsilon_4| \leq 2^{-t-1}$

$fix^*(w_4x_1^2) \equiv w_4x_1^2 + E_4; |E_4| \leq 2^{-t-3}$

(e) $fix^*(w_5x_2^2) \equiv w_5x_2^2 + E_5; |E_5| \leq 2^{-t-3}$

$$fix[f(x_1, x_2)] \equiv fix^*[f(x_1, x_2)] + \varepsilon_1 \equiv$$

$$f(x_1, x_2) + E_3 + E_4 + E_5 + \varepsilon_1$$

or,

$$fix[f(x_1, x_2)] \equiv f(x_1, x_2) + E$$

where

$$|E| = |E_3 + E_4 + E_5 + \varepsilon_1| \leq |E_3| + |E_4| + |E_5| + |\varepsilon_1|$$

$$\leq 2^{-t-4} + 2^{-t-3} + 2^{-t-3} + 2^{-t-1}$$

$$\leq 2^{-t}$$

Accumulation of Rounding Error -- It has been shown that the rounding error for one six-term element is bounded by 2^{-t-1} in the ideal case and 2^{-t} in the near-ideal case. When these elements are nested, the rounding error induced by the innermost elements (i.e., those in the first "layer" when the nest is expressed in a diagram) becomes input error to the next-innermost elements (those in the next layer), and so forth.

The effects of input error can be assessed by considering an arbitrary element with known input error. This can be done for the ideal case, the near-ideal case, or for other hardware realizations of an element.

The following analysis is for the near-ideal case; that is, the case in which the processor accumulates 2t-bit products, scales the 2t-bit element output by shifting left an appropriate number of bits (see Part 1(e)), and finally rounds the scaled product to t bits.

It is necessary to consider rounding error and propagation of input error simultaneously. Therefore,
Let

$$f(x_1, x_2) = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2$$

where:

$$|w_i| \leq 0.2 < 2^{-2} \text{ for } i = 1, 2, 4, 5$$

$$|w_j| \leq 0.1 < 2^{-3} \text{ for } j = 0, 3$$

$$|x_1| \leq 1, \; |x_2| \leq 1.$$

Let:

$$\bar{x}_1 = x_1 + \beta_1$$
$$\bar{x}_2 = x_2 + \beta_2$$

where:

$$|\beta_1| \leq 2^{-k}, \; |\beta_2| \leq 2^{-k}$$

$$k \leq t + 1$$

As in the previous section, E's will be used for rounding error.

(a) $fix^*(w_1\bar{x}_1) \equiv w_1\bar{x}_1 = w_1x_1 + w_1\beta_1 = w_1x_1 + E_1$

where $|E_1| = |w_1||\beta_1| \leq 2^{-2}2^{-k} = 2^{-2-k}$.

(b) $fix^*(w_2\bar{x}_2) \equiv w_2x_2 + E_2; \; |E_2| \leq 2^{-2-k}$

(c) $fix^*(w_3\bar{x}_1\bar{x}_2) \equiv w_3(\bar{x}_1\bar{x}_2 + \varepsilon_3) =$

$$w_3(x_1x_2 + x_1\beta_2 + x_2\beta_1 + \beta_1\beta_2) + w_3\varepsilon_3 =$$

$$w_3x_1x_2 + E_3$$

where:

$$|E_3| \leq |w_3x_1\beta_2| + |w_3x_2\beta_1| + |w_3\beta_1\beta_2| + |w_3\beta_3|$$

$$\leq 2^{-3}(2^{-k} + 2^{-k} + 2^{-2k} + 2^{-t-1})$$

$$= 2^{-3}(2^{-k+1} + 2^{-2k} + 2^{-t-1})$$

(d) $fix^*(w_4\bar{x}_1^2) \equiv w_4(\bar{x}_1^2 + \varepsilon_4)$

$= w_4(x_1^2 + 2x_1\beta_1 + \beta_1^2) + w_4\varepsilon_4$

$= w_4x_1^2 + E_4$

where:

$|E_4| \leq 2^{-2}(2{:}2^{-k} + 2^{-2k} + 2^{-t-1})$

$= 2^{-2}(2^{-k+1} + 2^{-2k} + 2^{-t-1})$.

(e) $fix^*(w_5\bar{x}_2^2) \equiv w_5x_2^2 + E_5$;

$|E_5| \leq 2^{-2}(2^{-k+1} + 2^{-2k} + 2^{-t-1})$.

Therefore

$fix[f(\bar{x}_1, \bar{x}_2)] \equiv fix^*[f(\bar{x}_1, \bar{x}_2)] + \varepsilon_1; \ |\varepsilon_1| \leq 2^{-t-1}$

$= f(x_1, x_2) + E_1 + E_2 + E_3 + E_4 + E_5 + \varepsilon_1$

$= f(x_1, x_2) + E'$

where

$|E'| = |E_1 + E_2 + E_3 + E_4 + E_5 + \varepsilon_1|$

$\leq 2^{-2-k} + 2^{-2-k} + 2^{-3}(2^{-k+1} + 2^{-2k} + 2^{-t-1}) +$
$2 \cdot 2^{-2}(2^{-k+1} + 2^{-2k} + 2^{-t-1}) + 2^{-t-1}$.

$= (2^{-k-1} + 2^{-k-2} + 2^{-2k-3} + 2^{-k} + 2^{-2k-1}) +$
$+ (2^{-t-4} + 2^{-t-2} + 2^{-t-1})$.

Therefore $|E'| \leq 2^{-k+1} + 2^{-t}$.

In this case, since $K \leq t + 1$, $2^{-k+1} \geq 2^{-t}$, and $|E'| \leq 2^{-k+2}$.

This result says, loosely speaking, that if an element operates on data with K-bit accuracy, the element output will be accurate to at worst K-2 bits, or less.

Part 3 - Word Length Requirements

If we assume for the moment that the input data to the nested polynomial are exactly represented with v bits, then we know (from Part 1) that 4 + v bits are needed to accommodate scaling and (from Part 2) that an extra 2 bits per "layer" (except for the first layer) are needed to prevent the propagation of rounding error through successive layers. Thus, for a nested polynomial of j + 1 layers, a word length

$$b = 4 + v + 2j$$

is required to maintain v-bit accuracy (i.e., error bounded by 2^{-v-1}), given near-ideal hardware.

Note that in general, the input data are not exact v-bit numbers, but are v-bit approximations to the true (usually unknown) numbers. Therefore, v-bit accuracy of the computed result of a nested polynomial cannot be guaranteed (regardless of the wordlength) because of the error inherent in the input data, which is compounded in successive computations. However, re-examining the total error (rounding error and propagation of input error) for a single element shows that

$$\text{fix}[f(\bar{x}_1, \bar{x}_2)] \equiv f(x_1, x_2) + E',$$

where (from Part 2)

$$|E'| \leq 2^{-k+1} + 2^{-t}.$$

That is, the error is made up of two components: rounding (2^{-t}) and input error propagation (2^{-k+1}). If the wordlength, t, is sufficiently large, the input-error-propagation term dominates. In fact, if we use the sharper inequality

$$|E'| \leq (2^{-k-1} + 2^{-k-2} + 2^{-2k-3} + 2^{-k} + 2^{-2k-1}) + (2^{-t-4} + 2^{-t-2} + 2^{-t-1})$$

and make the assumptions that $k \geq 5$ and $t \geq k + 2$, then:

$$|E'| \leq 2^{-k} + 2^{-k-1} + 2^{-k-2} + 2^{-k-3} + 2^{-k-4} + 2^{-k-5} + 2^{-2k-1} + 2^{-2k-3}$$
$$\leq 2^{-k+1}.$$

Therefore, setting $K = 4 + v$, we see that as long as our word length, t, is greater than $4 + v + 2 = 6 + v$, that our "per layer" error is only one bit.

In other words, for a nested polynomial having j layers, with input data consisting of v-bit approximations to the true (usually unknown) values, and implemented on a near-ideal processor with a word length, b, where

$$b \geq 6 + v$$

the maximum error, E', where

$$fix[f(\bar{x}_1, \bar{x}_2)] \equiv f(x_1, x_2) + E'$$

is bounded by

$$|E'| \leq 2^{-v-1+j}$$

7.3 SUMMARY

It has been shown that fixed-point arithmetic can be used for nested polynomial modeling.

The model training algorithm must be modified such that:

- The data base is appropriately scaled.
- The element coefficients satisfy certain inequalities.
- The intermediate variables are scaled.

Of these modifications, only the second limits the capability of the algorithm. This limitation can be made arbitrarily less restrictive by increasing word length.

For models consisting of 6-term elements, where the data base consists of v-bit approximations, the near-ideal processor having a word length of at least 6 + v bits has a combined error (roundoff and propagation) maximum of one bit per layer.

SECTION VIII

SIMULATIONS OF NONLINEAR MULTINOMIAL NETWORKS USING FIXED-POINT CPS's WITH VARIOUS WORD LENGTHS

8.1 SUMMARY

The major objective of the work covered in this section was to investigate via digital computer simulations the generation and propagation of errors caused by use of fixed-point arithmetic with finite word length data and finite arithmetic operations in nonlinear multinomial networks. Subsidiary objectives were to:

- Investigate the advantages of "near-ideal" or double-precision element computation.
- Study the effects of intermediate element scaling by shifting.
- Verify the theories proposed in the previous section with respect to worst-case error analysis for a fixed-point network.
- Determine the dynamic range of a fixed-point network as a function of both word length and the number of network layers.

The major conclusions reached in this section are:

- The theoretical derivation in Section VII is substantially confirmed by the simulation results, which reveal less propagation of errors than anticipated.
- A four-layer network will yield an accurate dynamic range of three decimal digits with 16-bit fixed-point computation.
- By employing 8-bit words, a network having two (and perhaps up to four) layers could accurately resolve a binary solution.
- "Near-ideal" element computation can improve network accuracy as much as 23 decibels (more than one decimal digit), compared to standard single-precision solutions, with little or no added hardware.
- Intermediate element scaling does not improve the dynamic range of network outputs.

8.2 SIMULATION PROCEDURE

8.2.1 NETWORK STRUCTURE AND SIZE

A four-layer "triangular" network having 15 elements and 16 unique inputs was used throughout the course of the simulation. Figure 43 illustrates this structure. Note that the output of any element never feeds more than one element in the next layer. This triangular configuration was selected on the basis of generality. In fact, any feed-forward network, no matter how intricate its connectivity, can always be expanded to the triangular form.

Each of the elements was a six-term quadratic function of two inputs expressed as:

$$y = w_o + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2$$

where x_1 and x_2 were the element inputs, w_0 through w_5 were the weighting coefficients, and y was the element output. All weighting coefficients and network inputs were selected at random.

8.2.2 GENERATION OF WEIGHTING COEFFICIENTS AND INPUT VALUES (W's AND X's)

The weighting coefficients were bounded by the following limits:

$$-.1 < w_0 < +.1$$
$$-.2 < w_1 < +.2$$
$$-.2 < w_2 < +.2$$
$$-.1 < w_3 < +.1$$
$$-.2 < w_4 < +.2$$
$$-.2 < w_5 < +.2$$

where the inputs never exceeded the domain

$$-1.0 < x < +1.0$$

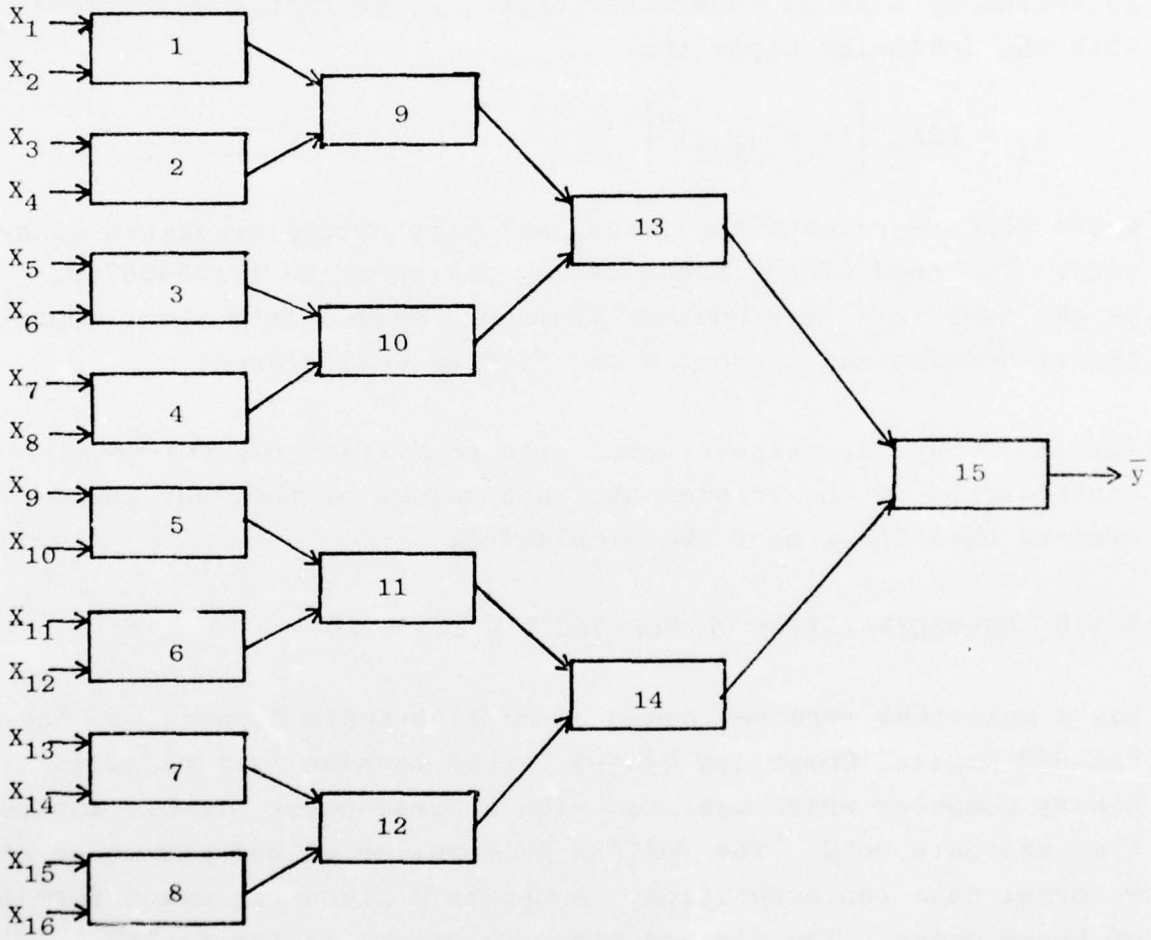


FIGURE 43: STRUCTURE OF THE SIMULATED NETWORK

These restrictions are those prescribed in Section 7, and will always guarantee fractional element outputs in the range

$$-1.0 < y < +1.0$$

prior to intermediate scaling of the y's.

The w's and x's were uniformly distributed psuedo-random numbers, generated by a HP-25 (Hewlett-Packard) programmable calculator with the following algorithm:

$$\mu_i = \text{FRAC}\left[(\pi + \mu_{i-1})^5\right]$$

where FRAC represents the fractional part of the bracketed quantity. The seed of the sequence, μ_o was equal to +.123456789. As the numbers were produced, they were mapped into their appropriate domains and truncated to five decimal places.

Tables 10 and 11, respectively, give the values for the weighting coefficients of the 15 elements, and values of the four input vectors used throughout the simulation.

8.2.3 CHARACTERISTICS OF THE DIGITAL COMPUTER

The simulations were performed on an Electronic Associates, Inc. EAI-640 Digital Computing System. This machine is a parallel binary computer which operates with a fixed-point, 16-bit instruction and data word. The FORTRAN IV compiler allows six modes of internal data representation. Appendix H gives the exact format of these modes. The x's and w's were stored in the Scaled Fraction mode, allowing for a 15-bit mantissa, plus sign bit.

A convenient feature of the compiler was the allowance of "In-Line Assembly Coding," permitting both FORTRAN and assembly language statements to be interspersed throughout the program. FORTRAN

TABLE 10

RANDOM NETWORK COEFFICIENTS FOR THE 15 ELEMENTS

ELEMENT	w_0	w_1	w_2	w_3	w_4	w_5
1	-.08721	.14660	.02676	-.00363	-.01220	-.19320
2	.03765	-.05703	.00134	.09677	.04519	-.19662
3	-.09765	-.15426	.15460	.06271	-.01232	.07394
4	.08773	-.08215	-.17340	-.09078	.12359	-.06283
5	-.07302	.10504	-.14389	.03109	.08444	.10183
6	-.05569	.14108	.12103	-.07058	.01190	-.17672
7	.03176	-.05493	-.01419	.06967	.12957	.08813
8	-.06631	-.19671	-.19857	-.04150	.10168	-.01269
9	.00057	.18045	-.08032	.05383	-.09912	-.11364
10	.04821	.02874	-.12869	-.04812	-.10702	-.09594
11	.09918	.01007	.11724	.01608	-.11074	-.12133
12	-.04791	-.00900	.15228	.03347	.19409	.15545
13	.02722	-.04660	-.07095	-.01275	.16305	.03170
14	-.05676	-.11660	.16220	.04769	.08697	.12951
15	.00390	.01510	-.06530	.04293	-.15029	.00854

TABLE 11

THE FOUR RANDOM INPUT VECTORS USED IN SIMULATION

Record	x_1, x_5, x_9, x_{13}	x_2, x_6, x_{10}, x_{14}	x_3, x_7, x_{11}, x_{15}	x_4, x_8, x_{12}, x_{16}
1	-.26338	.38124	.0915	.74762
	.73512	.53310	.40036	.73922
	-.16192	-.37131	.96073	.83810
	-.97022	-.31728	.10235	.37298
2	.86365	.39380	-.38857	.71173
	.50089	-.86367	.38234	-.79538
	-.59514	-.67376	-.68517	.52935
	.03540	-.75119	.20782	-.72102
3	-.47227	-.97145	-.92705	-.59671
	.34433	.59799	-.64500	.61461
	-.51813	.59595	.89486	-.73471
	-.37323	-.40664	.01613	.06401
4	-.90303	-.25230	.81302	-.86339
	.53361	.98970	.95420	-.56541
	.08380	-.78364	-.07408	-.01327
	.18673	.64049	-.85137	.93473

was used as much as possible; however, it was necessary to write a few assembly language routines, including double precision add and multiply systems programs and variable bit mask programs to perform the "near-ideal" method of element computation at various word-lengths. A listing of the 640 instruction repertoire is given in Appendix H.

8.2.4 COMPUTATION OF THE "NEAR-IDEAL" ELEMENT

A method which greatly minimizes the effects of multiplication roundoff error during the computation of a six-term element has been devised. It has come to be named "near-ideal" element computation because its employment produces "near-perfect" element outputs when compared to their true values. In essence, the method simply performs the multiplication and addition of the x's and w's using double-precision arithmetic. After the six terms are summed, the double-precision result is rounded to single precision.

A result of these simulations shows that near-ideal element computation reduces the average absolute error as much as 23 decibels (more than one decimal place!) in the network, when compared to standard single-precision element computation, where rounding takes place after every multiplication.

Figure 44 illustrates how "near-ideal" elements were computed in this study. The letter t represents the word length in bits of the x's and w's. A multiplication of two t-bit quantities produces a 2t-bit result.

Instead of rounding the products of the w_1x_1 and w_2x_2 terms, their exact products were accumulated in a double-precision (2t-bit) summing register. For the terms involving two multiplications ($w_3x_1x_2$, $w_4x_1x_1$, and $w_5x_2x_2$), a 3t-accumulator and a 2t-by-t multiplier would be required to represent exactly

w_i - weighting coefficients

x_1, x_2 - input variables

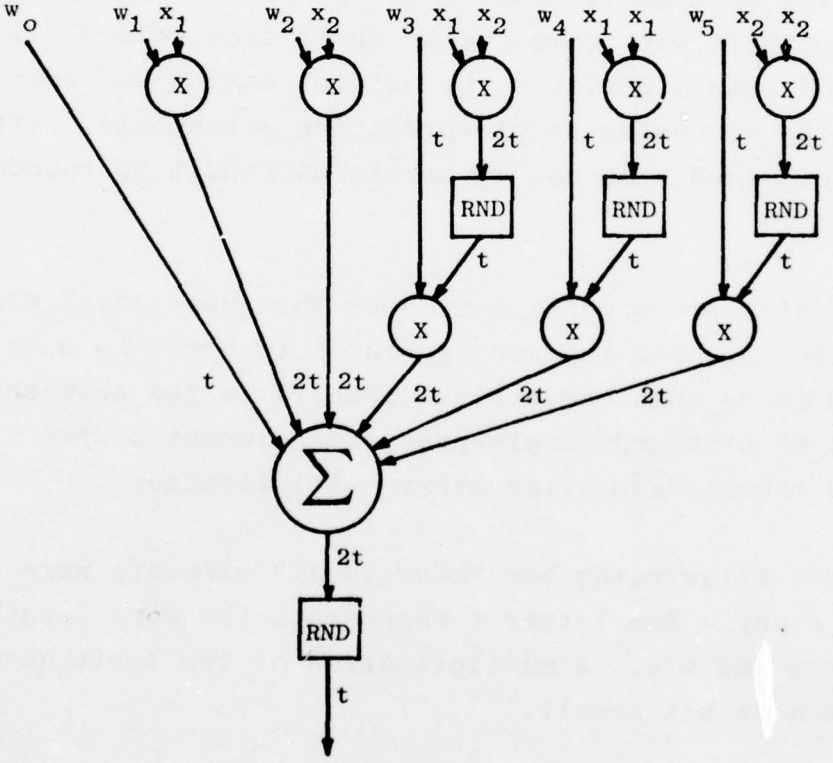
t - number of bits precision

y - element output

(X) - multiply operation

RND - rounding operation

(Σ) - summation operation



$$y = w_o + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_1x_1^2 + w_5x_2^2 *$$

*Factored forms of this equation are alsc possible but were not considered in thes simulations.

FIGURE 44: OPERATIONS REQUIRED TO PERFORM "NEAR-IDEAL" ELEMENT COMPUTATI

these products. Instead, in the "near-ideal" case, the partial term products x_1x_2, x_1x_1, and x_2x_2 were rounded to t bits, then multiplied by their appropriate coefficients to produce 2t-bit products. These double-precision quantities were then accumulated in the summing register. The t-bit w_0 was directly accumulated. Following the addition of the six terms, the double-precision result was rounded to t bits, yielding the "near-ideal" element output.

The rounding operations were performed by adding one to the (t+1)-bit position of the double precision quantities, then truncating the result to t bits.

8.2.5 REPRESENTING FEWER THAN 16 BITS WITH A 16-BIT COMPUTER

One objective of this study was to investigate errors produced in the network as a function of computational word length. It was therefore necessary to simulate arithmetic operations for various specified bit precisions. The word lengths selected were 16, 14, 12, 10, 8, and 6 bits.

These word lengths were simulated by masking, (or truncating), the x's and w's before and after they were involved in an arithmetic operation. The double-precision product formed in a multiplication operation was masked to 2t bits.

8.2.6 COMPOSITE AND COMPUTATIONAL ERRORS DERIVED FROM TRUE VALUE

To compute the error generated in the network as a result of finite word length arithmetic operations, a "true" value for each of the 15 elements was computed. Using the double-precision mode of the EAI-640, the 16-bit scaled-fraction x's and w's were converted to 62-bit floating-point values (53-bit mantissa plus sign and 7-bit exponent plus sign). A separate subroutine was written to compute the entire network in the floating-point, high-precision

mode, while another section of the program computed the network using short-mantissa, fixed-point arithmetic to the specified number of bits. A comparison of the fixed-point element outputs to the double-precision element outputs indicated the degree of error introduced by the fixed-point operations.

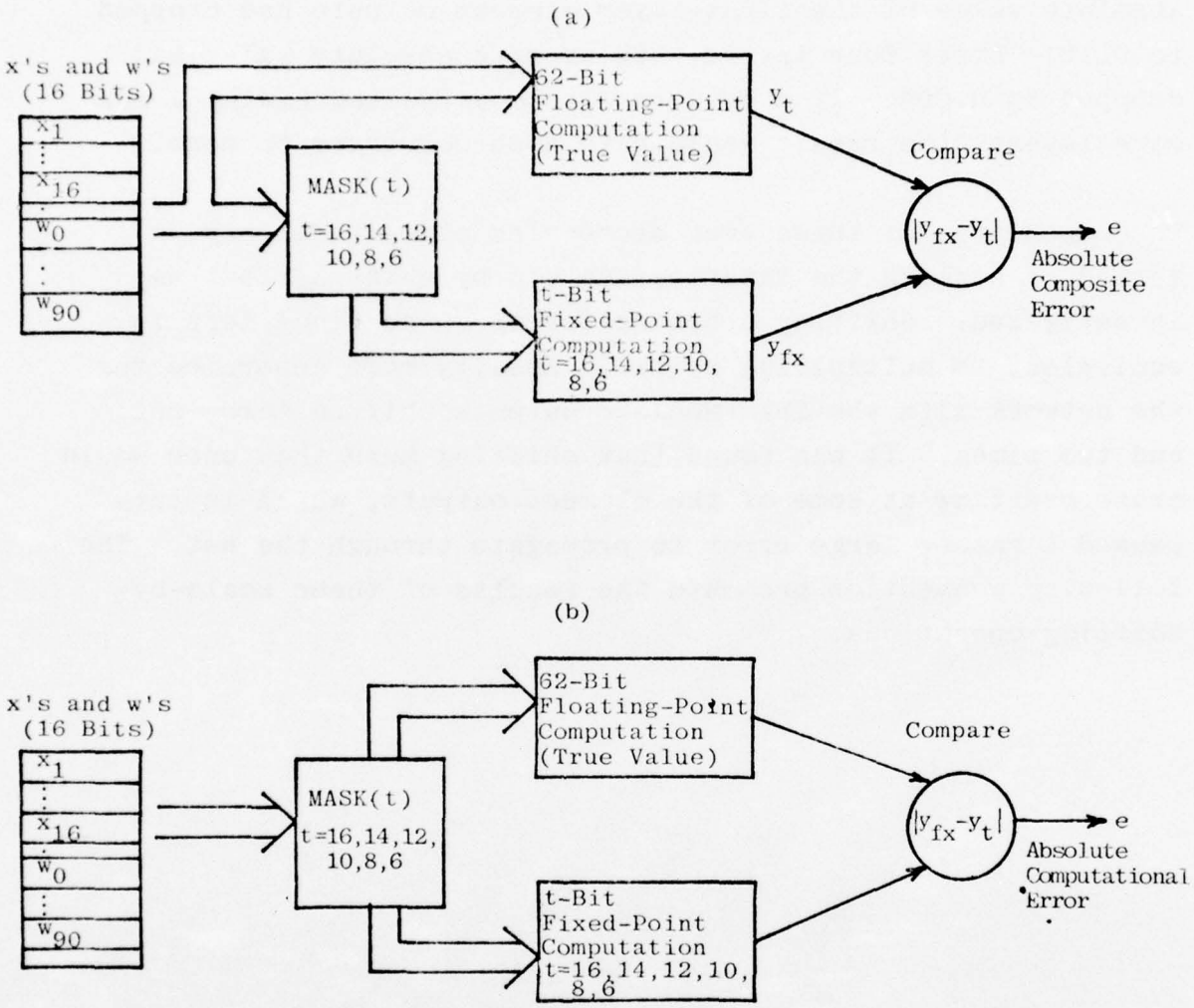
Two types of errors can be generated and propagated through the network -- those caused by round-off after an arithmetic computation, and those due to finite representations of the input data to the network. The former will be termed "computational" error, and the latter "input data" error. The name "composite" error will be given to those quantities containing both computational and input data errors.

Both the x's and w's are received with a certain degree of error, from the point of view that these digital quantities are approximate representations of continuous functions.

Figure 45 shows the procedure used for finding the composite error and the computational error in the fixed-point network. Notice, in the upper diagram, that the input data (x's and w's) used to calculate the true value remain fixed at 16 bits when the input data to the fixed-point network are masked to t bits. In the lower diagram, notice that the input data to both the fixed-point and 62-bit floating-point networks are always accurate to the same number of bits.

8.2.7 INTERMEDIATE ELEMENT SCALING

An inherent restriction imposed on the network when using fixed-point arithmetic is that the coefficients of the elements must be limited to a fractional range. The theoretical bounds of these coefficients are established in Section VII. Although these bounded coefficients guarantee the element outputs to be fractional, their over-all effect is to decrease the dynamic range



y_{fx} - fixed-point element outputs

y_t - "true" element outputs

FIGURE 45: PROCEDURE FOR FINDING (a) COMPOSITE ERROR AND (b) COMPUTATIONAL ERROR

of the element inputs. In one example, the average absolute value of the x's inputted to the network was 0.50. The average absolute value of the first-layer element outputs had dropped to 0.12. After four layers, the average absolute value had dropped to 0.009. If a network were constructed having a few more layers, the result would have soon converged to zero.

To compensate for these ever decreasing element outputs, a method of scaling the intermediate y's by shifting left was investigated. Shifting a binary fraction one place left is equivalent to multiplying by two. Results were generated for the network with the intermediate outputs shifted zero, one, and two times. It was found that shifting more than once would cause overflow at some of the element outputs, which in turn caused a rather large error to propagate through the net. The following subsection presents the results of these scale-by-shifting operations.

8.3 SIMULATION RESULTS AND ERROR ANALYSIS

This section presents a graphical display of the average and maximum errors generated and propagated in the simulated fixed-point networks, where the average error for a given layer was calculated for the ensemble of outputs from that layer.

Figures 46 (a-f) show the average absolute composite error and maximum absolute composite error[1/], as a function of computational word length for three cases of intermediate scaling: shift = 0 (no scaling), shift = 1, and shift = 2. Figures 46 (g-l) illustrates average absolute computational error and maximum absolute computational error as functions of computational word length for the same three shift values. Each individual figure shows the average absolute error (either composite or computational) observed after each layer in the network (solid lines), and the average absolute value observed after each layer (dashed lines). The terms L=1, L=2, etc., refer to the layer numbers for the average absolute values. All twelve plots were derived from a network employing "near-ideal" elements.

The number of available data points from which the plots in Figure 46 were derived varied from layer to layer. The number of points available was:

Layer	Number of Data Points
1	32
2	16
3	8
4	4

The variation was due to the triangular structure of the net. It follows that the statistical significance of the simulation results is greater for layer 1 and diminishes with each successive layer.

1/ The maximum absolute error was the largest error observed for a given simulation trial.

The purpose for plotting the average absolute value on the same figure with the absolute errors is to indicate the approximate dynamic range of the element outputs for the various layers. Notice for all the curves in Figure 46, except those labeled "shift =2", that the error at the least significant portion of y is a function of the computational word length, but the average absolute value at the most significant portion of y is a function of the layer. The approximate dynamic range of y is equal to the average absolute value minus the absolute error. For example, from Figure 46 (g) it is seen that with 16 bits of computational word length the average dynamic range for a four-layer network is about 60 decibels, or three decimal digits. The same network computed with 8-bit words yields about 10 decibels for the average dynamic range. This is better illustrated in Figure 47, where the average dynamic range observed after each layer has been plotted as a function of the computational word length. This figure has been derived from Figure 46 (g) by subtracting the average absolute error from the average absolute value. Notice that as the number of layers increases, the dynamic range of the network decreases. A similar plot could be derived from the maximum absolute error versus word length to find the "guaranteed" dynamic range.

The effect of shifting the element outputs one place left is shown in the middle column of Figure 46. (Those figures labeled "shift = 1".) Although this scaling procedure increased the average absolute value of the element outputs, it also increased the error by an equally proportionate amount. The magnitude of the number is multiplied by a factor of two, but the error is also moved into a more significant bit position, causing a zero net increase of dynamic range. Intermediate element scaling is therefore not necessary unless the layer outputs are required to maintain a fixed numerical value.

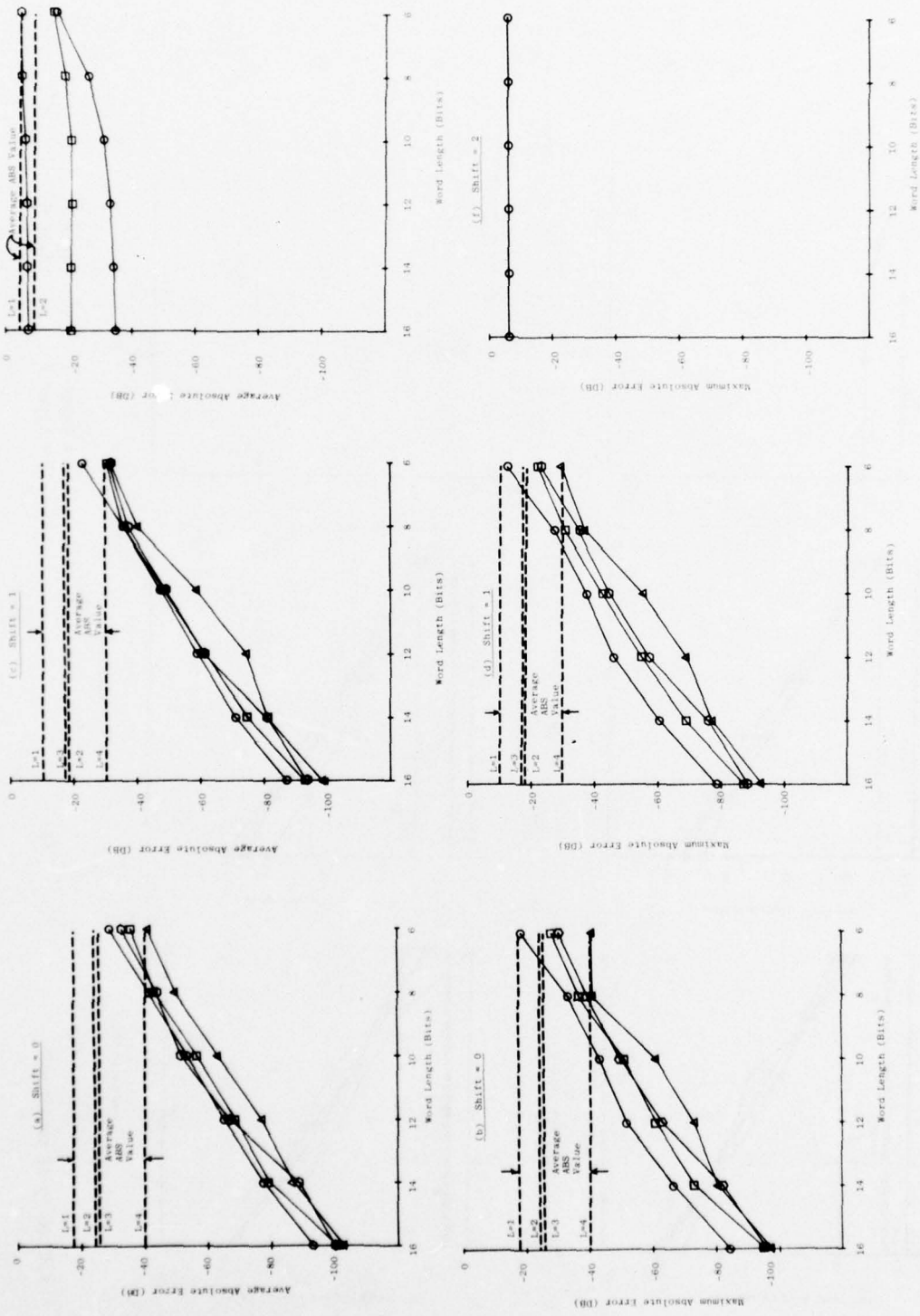


FIGURE 46 (a-f) AVERAGE AND MAXIMUM ABSOLUTE ERRORS VERSUS COMPUTATIONAL WORD LENGTH FOR THREE CASES OF INTERMEDIATE SCALING

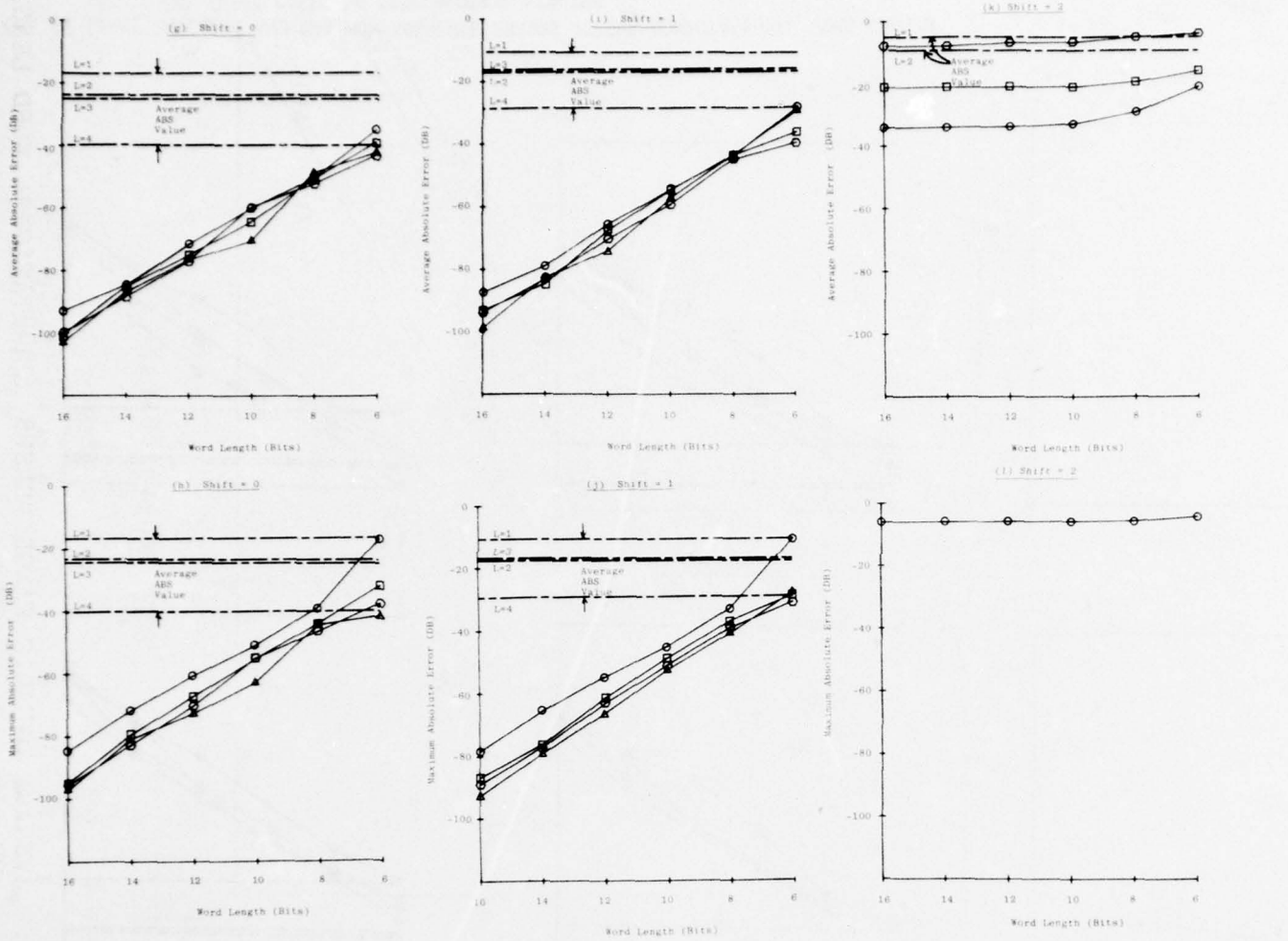


FIGURE 46 (g-1) continued:

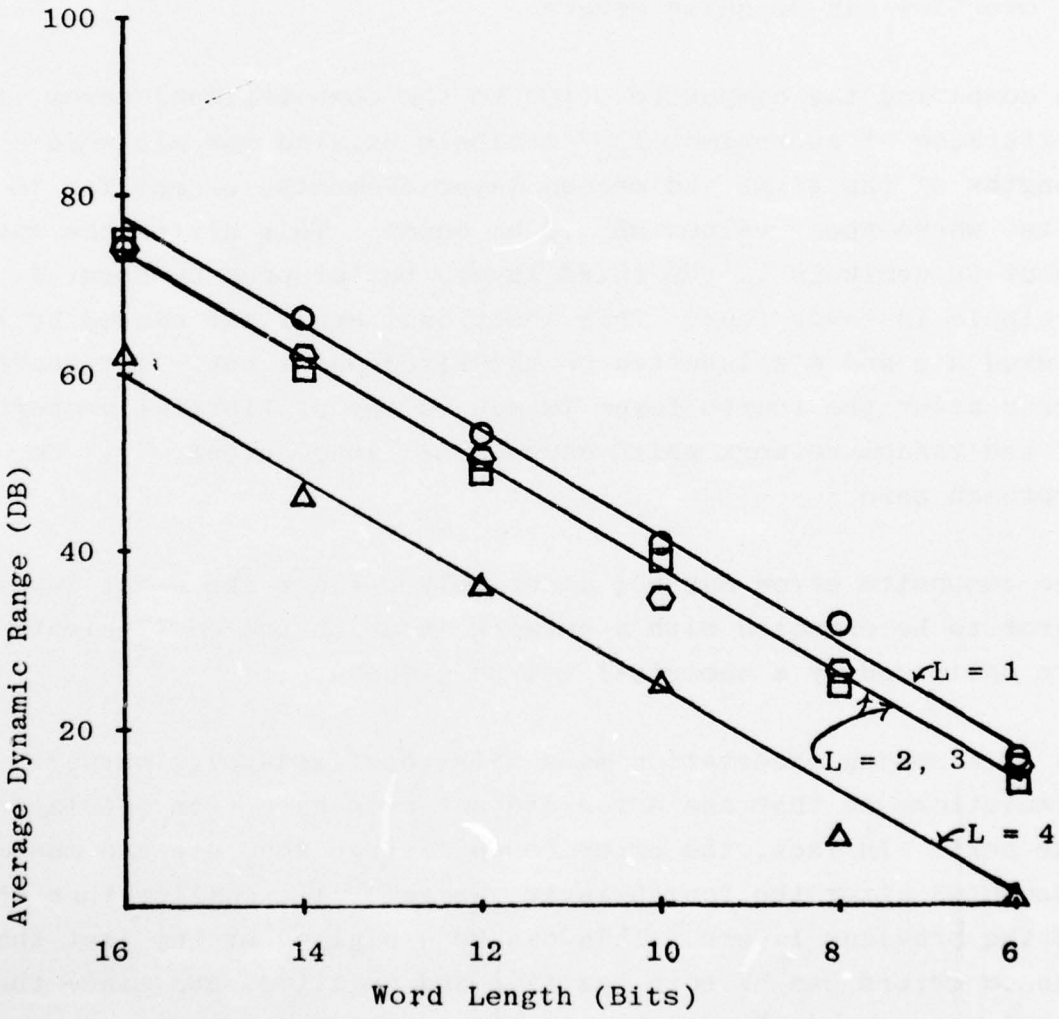


FIGURE 47: AVERAGE DYNAMIC RANGE VERSUS COMPUTATIONAL WORD LENGTH FOR SIMULATED NETWORK

The results in the third column of Figure 8.4 (labeled "shift = 2") illustrate the effects of overflow error propagation. In these figures, all network outputs were shifted two places left (equivalent to multiplying by four) before being processed through the next layer. This scaling was too large for some of the intermediate outputs, causing them to be saturated at 0.9999, hence, large errors were propagated. The guaranteed dynamic range of the network thus dropped below zero decibels, as indicated by Figures 8.4 (f and l). As seen by these diagrams, the effects of overflow can be quite severe.

In comparing the composite error to the computational error, a difference of approximately 7 decibels existed for all word lengths of the first and second layer elements, except for 16 bits, where these values should be equal. This difference was about 10 decibels in the third layer, but dropped to about 3 decibels in layer four. This additional error was caused by the masked x's and w's inputted to the fixed-point net. The decreased error after the fourth layer is due to the statistical property of the random network, which caused the fixed output, "y," to approach zero.

The composite error may not accurately reflect the exact level of error to be expected with a network in which the coefficients are optimized by a numerical search procedure.

An interesting observation made with the fixed-point network simulations is that the error did not propagate from one layer to the next. In fact, the error (both average absolute and maximum absolute) after the fourth layer was generally smaller than that of the previous layers. This can be explained by the fact that random errors can be both positive and negative, and since the final network output represents the summation of many thousands of terms, the probability that these errors cancelled one another is high. Also, when using fixed-point arithmetic, there are no

exponents in the numbers to automatically scale errors to higher significant positions. The errors therefore tend to remain confined to the least significant bits.

As further verification of the findings of this study, the w's and x's were all given positive polarities in additional trials to assure that the network output would be a positive number. The results were essentially identical to those just described, except that the first layer errors were about 6 decibels larger for the composite errors, and about 3 decibels larger for the computational errors (compared with the result shown in Figure 8.4). Also, with input x's being positive, the element outputs were larger in magnitude and, hence, shifting could not be performed without overflow.

Figure 8.6 shows the theoretical maximum composite error as a function of word length for each layer in a four-layer network. This diagram was produced from the equation shown on the figure which was derived in Section 7. The theoretical error for layer one is almost identical to the maximum computational error observed in the simulations (Figure 8.4-h), but is about five decibels (one bit) less than the observed maximum composite error (Figure 8.4-b). This difference of about one bit is believed to have been caused by the computer system software when converting input data from scaled-fraction to double-precision mode. Consequently, the observed first layer results of the figures showing computational error (Figures 8.4, g-l) may be (at most) one bit larger than the actual error.

It is also seen from Figure 8.6 that the composite error increases as the number of layers increases, due to the propagation of the feed-forward input error. The maximum error propagation is shown to be 21 decibels between the outputs of Layer 1 and Layer 4. The simulation results have indicated, however, that error cancellation occurs if the computational errors distribute themselves in a random fashion. Hence, the actual error propagation between layers can be considerably less than the expected worst case.

This figure was derived from the equation $|E'| \leq 2^{-k+1} + 2^{-t}$ on Page 116, and represents "near-ideal" computation.

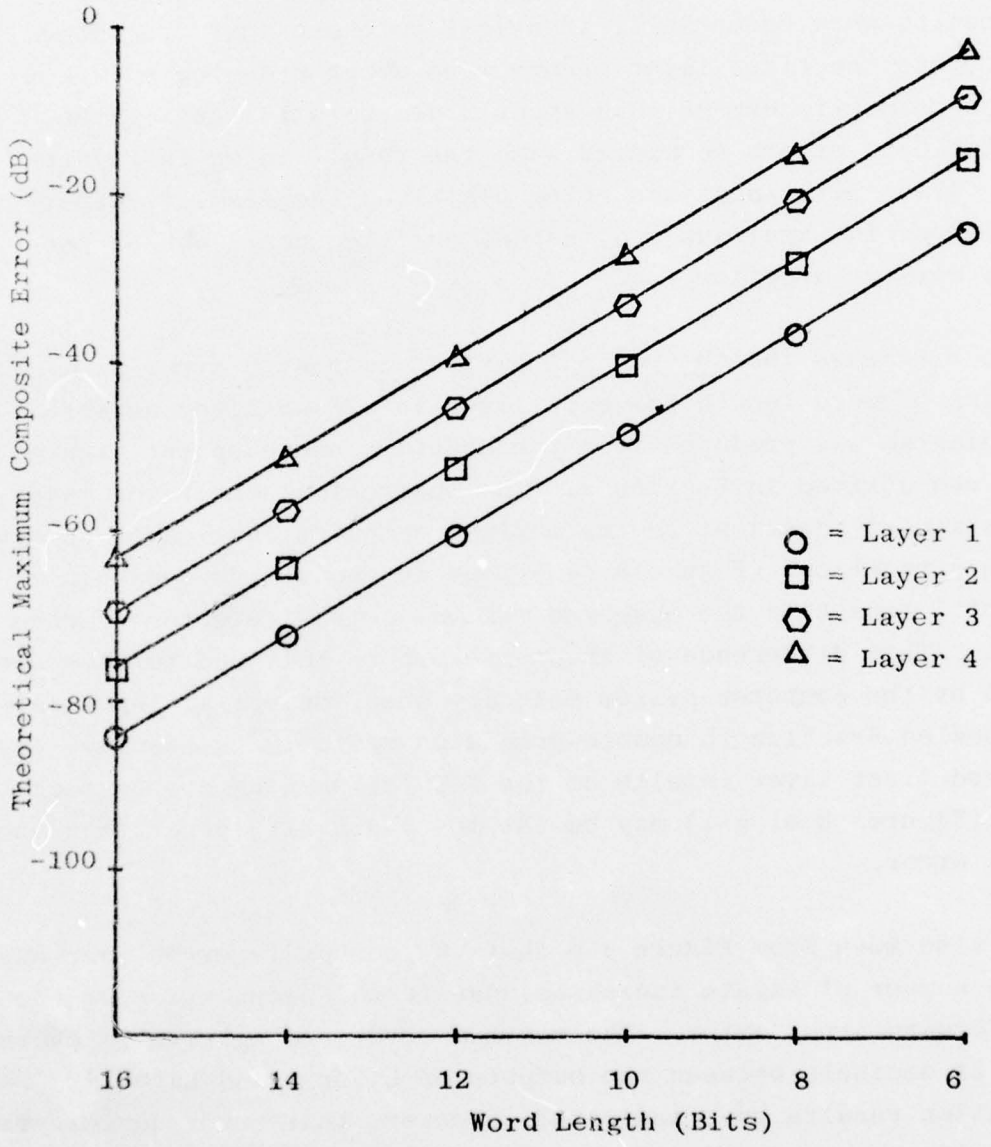


FIGURE 48: THEORETICAL MAXIMUM COMPOSITE ERROR FOR "NEAR-IDEAL" HARDWARE.

Since the theoretical derivations of Section VII are based on a "worst case" analysis, it is not surprising to see a somewhat sizeable difference between the theoretical and observed maximum errors of the second, third, and fourth layers. For a network having non-random coefficients, and with non-random inputs, it is not known how the errors will distribute themselves, but, such a network should be a prime consideration in future simulations. In comparison to the random network studied herein, the theoretical findings of Section VII are seen to be conservative with respect to error propagation.

8.4 CONCLUSIONS AND RECOMMENDATIONS

The following conclusions have been reached as a result of the simulation described in this section.

- The theoretically-derived conclusions presented in Section VII have been substantially confirmed, except that in the simulated case the worst-case error propagation was not observed. It is felt that this more favorable outcome resulted from cancellation of random errors within the simulated network.
- For applications requiring only a binary discrimination, as in the case of a pairwise-voting classifier, eight bits would be sufficient for computing a two-layer (shallow) network, and perhaps a three- or four-layer network, but fewer than eight eight bits is not recommended for any application.
- A four-layer (moderately deep) network of six-term nonlinear multinominal elements has an accurate dynamic range of three significant decimal digits if computation is performed with 16-bit fixed-point "near ideal" arithmetic. Smaller networks yielding greater precision, or larger networks yielding less precision, are also possible with 16-bits. For applications requiring greater accuracy, a larger work length will be needed.
- Deep networks of up to eight layers should be realized using at least a 24-bit mantissa to obtain three significant decimal digits of resolution at the network output.
- "Near-ideal" element computation improves the network accuracy as much as 23 decibels. (Implementation of this type of arithmetic requires little or no additional hardware compared with the standard method of computation.)
- If no intermediate scaling is performed in a CPA network, the expected range of the element outputs decreases as the number of layers increases. More than 20 decibles of attenuation have been observed going from the first layer output of the network to the fourth layer output. However this decrease is not a great hindrance: it can be mitigated at the least significant portion of the element outputs by keeping error propagation at a minimum. In fact, the errors observed in the fourth layer outputs were almost always smaller than errors observed at the previous layer. (Without shifting, the computational error remains confined and is not moved into higher-order bit position.) It was found that the dynamic range, or number of significant figures of the element outputs, remained constant, with or without intermediate scaling.
- Propagation error caused by intermediate element overflow was observed to be quite severe on the final network output. Errors of this type tend to have a "snowballing" effect and should be avoided.

It is recommended that the simulation results be verified using simulations of a fixed-point network whose coefficients are selected, or "optimized," from training data. The statistical distribution of errors for such a case may not be as random as we have seen with the contrived network, and hence may cause a greater error propagation through the layers.

SECTION IX

LINEAR AND NONLINEAR FILTERING WITH CPS's

9.1 INTRODUCTION

The architecture of configurable polynomial arrays is very well suited for the implementation of a broad class of digital filters, both transversal and recursive. Procedures exist for implementing transversal filters, fast Fourier transforms impulse-response convolution filters, etc., with various LSI arrays. (See, for example, References 4 and 5.) This section presents procedures for implementing three common configurations of recursive linear filters with CPA's.

Section 9.3 presents a novel concept of trainable, nonlinear filters. In many signal recognition and classification problems, the key waveform features are nonlinear properties of the signal frequency spectrum. A complicating factor is that the nonlinear property may not be known a priori and must be "learned" from empirical data. Adaptive learning network (ALN) procedures may be used to synthesize an appropriate nonlinear filter from a data base. In turn, these filters are easily implemented by CPA's. By way of example, an ALN is trained to estimate a parameter which is known (to us) to be the energy within a certain frequency band of a signal. The resulting ALN algorithm is compared to a conventional frequency-band energy estimator to show that the accuracies and computational costs of trained ALN models are comparable to a priori conventional implementations even if the actual transformations are known.

In summary, configurable polynomial networks are appropriate for the implementation of conventional, linear filters (both transversal and recursive) but are also capable of implementing more general nonlinear filters. Adaptive learning network procedures may be used to synthesize the mathematical forms of the nonlinear filters from empirical data.

9.2 PROGRAMMABLE POLYNOMIAL NETWORK IMPLEMENTATIONS OF RECURSIVE DIGITAL LINEAR FILTERS

9.2.1 CONVENTIONAL STRUCTURES

Direct Form -- recursive linear digital filters may be written as transfer functions in the z-domain:

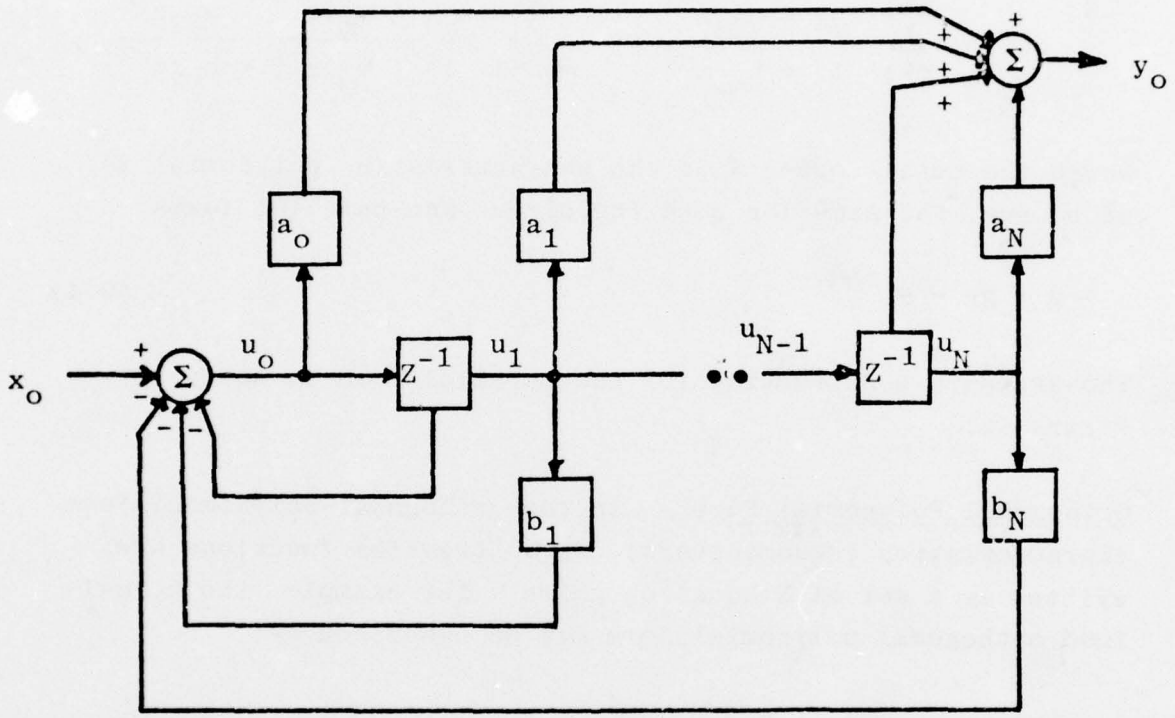
$$H(z) = \frac{y(z)}{x(z)} = \frac{a_o + a_1 z^{-1} + \ldots + a_N z^{-N}}{1 + b_1 z^{-1} + \ldots + b_N z^{-N}} \tag{9.1}$$

or

$$H(z) = \frac{\sum_{n=0}^{N} a_n z^{-n}}{1 + \sum_{n=1}^{N} b_n z^{-n}} \tag{9.2}$$

where z^{-1} is the delay operation, N is the order of the transfer function, and a_n and b_n are constant coefficients. This form of the filter equation is called the direct form. Hardware or software may be constructed, as shown in Figure 49 to implement this direct form.

In many applications it is desirable to implement the transfer functions in different forms so as to improve numerical accuracy and/or computational stability margin, which may be troublesome due to finite word lengths in digital computers. Two prominent alternate forms are the parallel and orthogonal polynomial forms



$$u_o = x_o - \sum_{n=1}^{N} b_n u_n$$

$$y_o = \sum_{n=o}^{N} a_n u_n$$

$$u_n = u_{n=1} Z^{-1} \qquad (n = 1 \cdots, N)$$

FIGURE 49: LINEAR DIGITAL FILTER, DIRECT FORM, STANDARD CONSTRUCTION

Parallel Form -- With the parallel form, Equation 9.2 is factored, by partial fractions, into a parallel sum of linear and quadratic transfer functions:

$$H(z) = \sum_{m=1}^{M} \frac{a_{m0} + a_{m1}z^{-1}}{1 + b_{m1}z^{-1}} + \sum_{m=M+1}^{M+L} \frac{a_{m0} + a_{m1}z^{-1} + a_{m2}z^{-2}}{1 + b_{m1}z^{-1} + b_{m2}z^{-2}} \quad (9.3)$$

where the total degree N of the characteristic polynomial is, of course, the same for both the direct and parallel forms:

$$M + 2L = N \quad (9.4)$$

The standard construction for the parallel form is shown in Figure 50.

Orthogonal Polynomial Form -- In the orthogonal polynomial form, characteristics (denominators) of the transfer functions are written as a set of N equation pairs. For example, the normalized orthogonal polynomial form may be described by:

$$u^{+}_{n-1} = c_n u^{+}_{n} - k_n u^{-}_{n-1} z^{-1}$$

$$u^{-}_{n} = k_n u^{+}_{n} + c_n u^{-}_{n-1} z^{-1} \qquad n = 1,N \quad (9.5a)$$

and

$$u^{+}_{N} = x$$

$$u^{-}_{o} = u^{+}_{o} \quad (9.5b)$$

where the k's and c's are constant coefficients and the u^+_n and u^-_n are intermediate variable pairs. The numerator portion of the transfer function is given by the polynomial:

$$y = \sum_{n=o}^{N} a_n u^-_n \qquad (9.5.c)$$

These equations (9.5.a, 9.5.b, and 9.5.c) give rise to the ladder structure shown in Figure 51.

9.2.2 Polynomial Network Structures

All polynomial representations of linear filters may be implemented in configurable polynominal networks. Net structures for the direct, parallel, and orthogonal polynomial forms of Equations 9.2, 9.3., and 9.5 are shown in Figures 52, 53 and 54, respectively. All calculations can be performed with two configurations of elements, here called P5 and P6. Both P5 and P6 are four-input elements; however, P5 has two outputs and P6 has only one:[1/]

$$\text{P5:} \quad \begin{cases} y_1 = w_1x_1 + w_2x_2 \\ y_2 = w_3x_3 + w_4x_4 \end{cases} \qquad (9.6)$$

$$\text{P6:} \quad y = w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 \qquad (9.7)$$

where: the y's are the element outputs, x's are the inputs, and w's are constant coefficients.

[1/] P6 is obtained from y_{1B} (Sections 1 and 2 of this report) by setting $w_o = 0$.

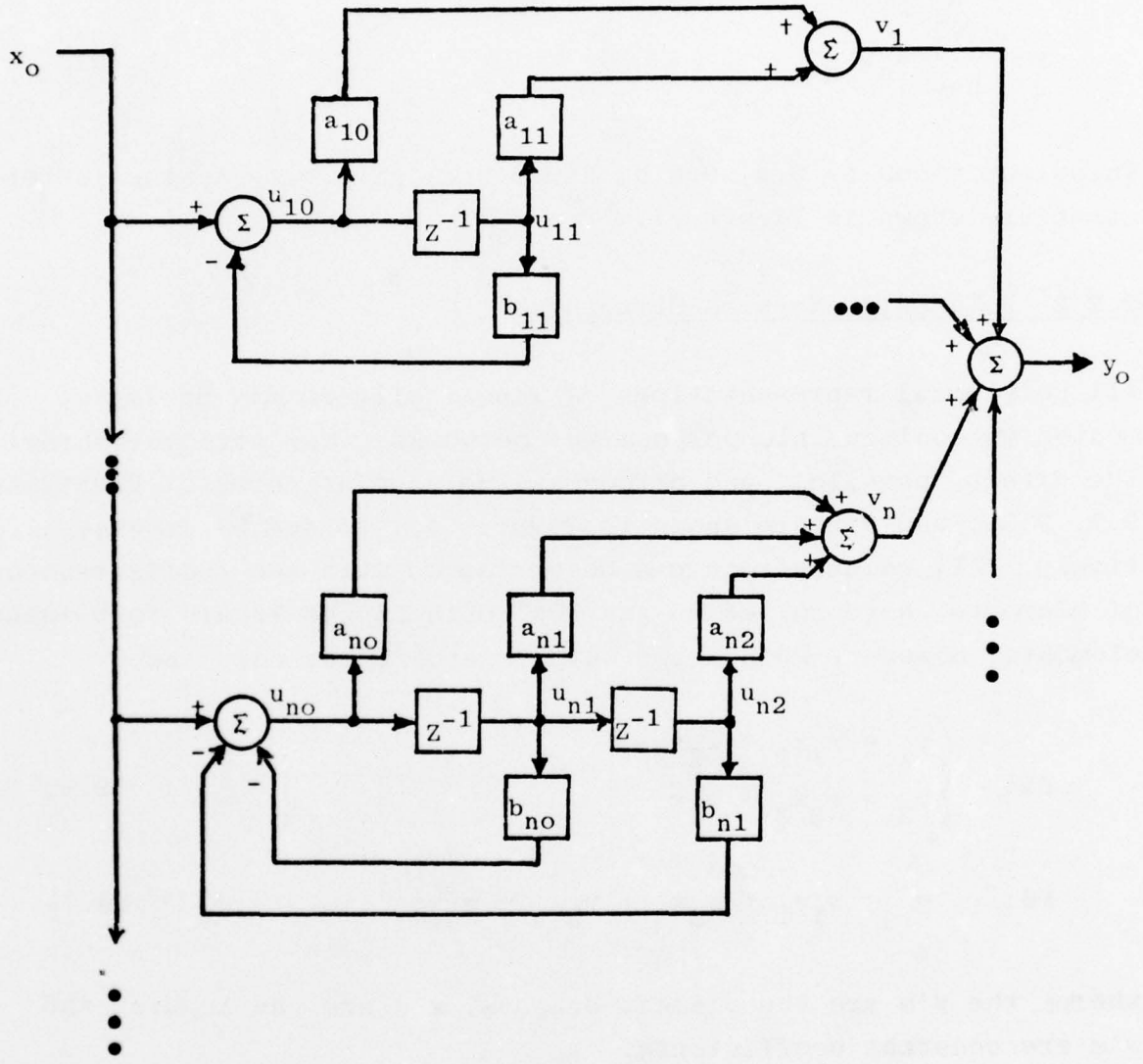


FIGURE 50: LINEAR DIGITAL FILTER, PARALLEL FORM, STANDARD CONSTRUCTION

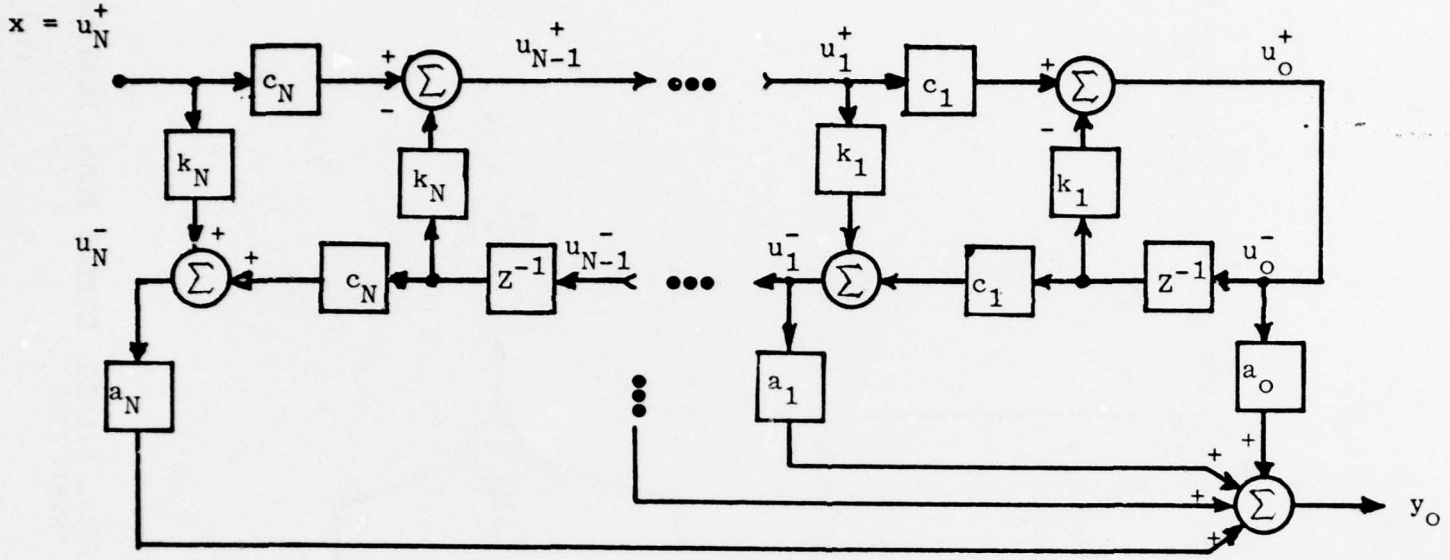


FIGURE 51: LINEAR DIGITAL FILTER, ORTHOGONAL POLYNOMIAL FORM, STANDARD CONSTRUCTION

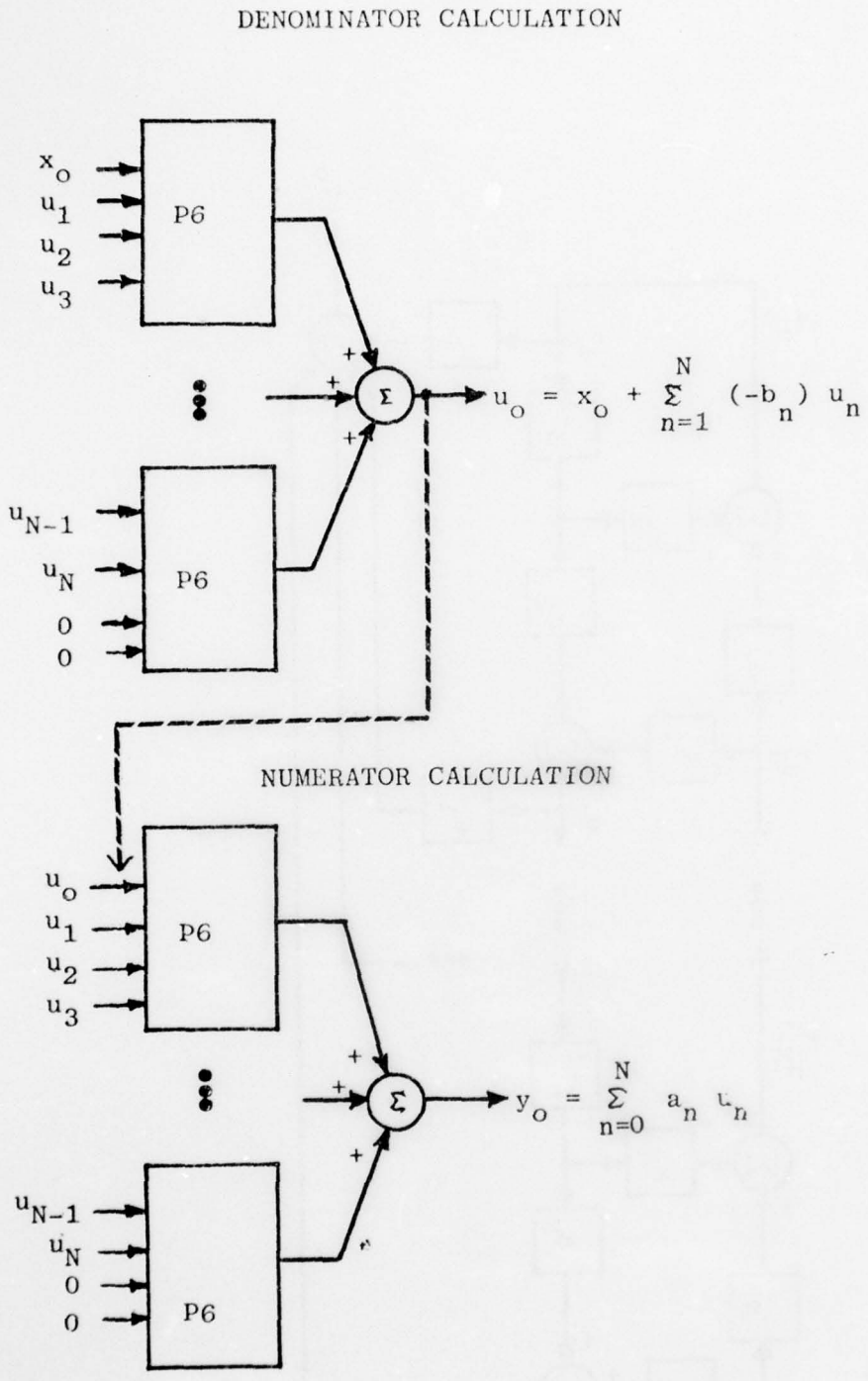


FIGURE 52: LINEAR DIGITAL FILTER, DIRECT FORM, POLYNOMIAL NETWORK CONSTRUCTION

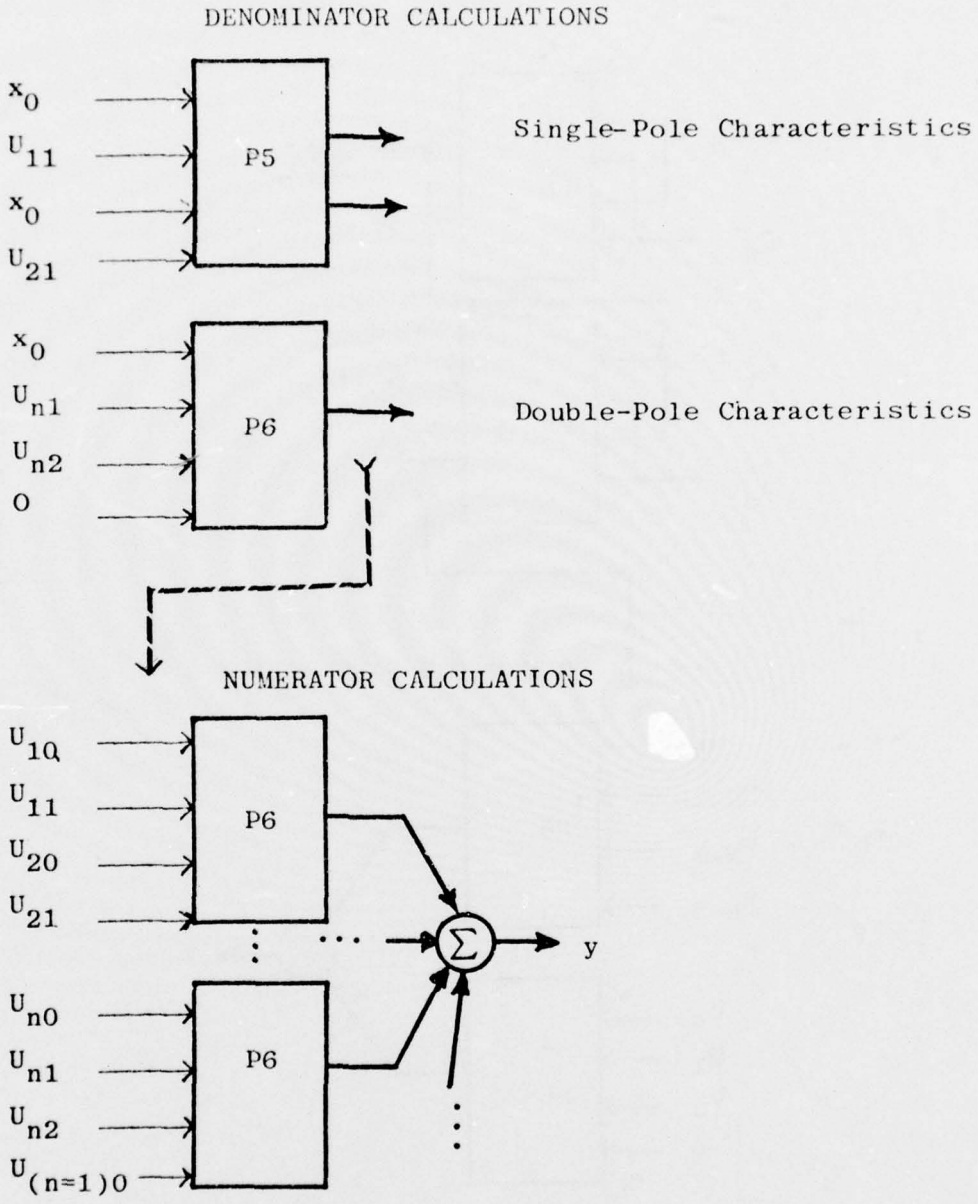


FIGURE 53: LINEAR DIGITAL FILTER, PARALLEL FORM, POLYNOMIAL NETWORK CONSTRUCTION

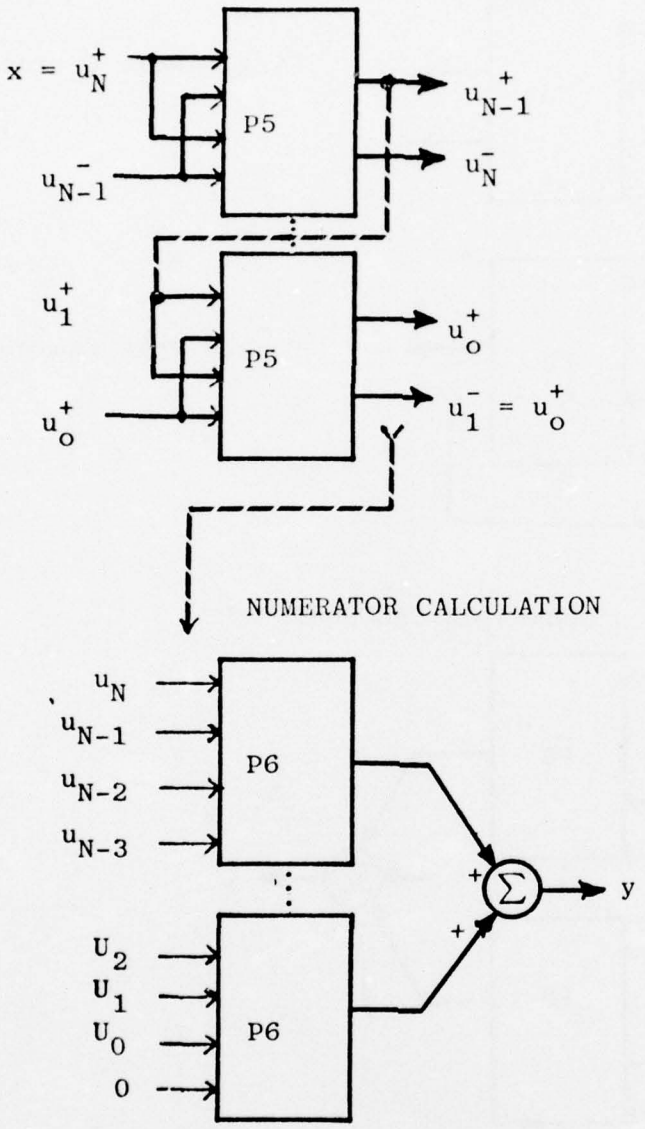


FIGURE 54: LINEAR DIGITAL FILTER, ORTHOGONAL POLYNOMIAL FORM, POLYNOMIAL NETWORK CONSTRUCTION

Number of Elements Required for Each Form -- The number of elements, E, required for an N-pole filter is generally as follows:

Direct Form:	$E_d = (N+1)/2$
Parallel Form:	$E_p = N$
Orthogonal Polynomial Form:	$E_o = N + (\frac{N+1}{4})$

For the direct form, an (N + 1)-term series must be computed for the numerator and for the denominator. Since four terms are summed per element (P6), each summation requires (N+1)/4 elements, so the total element count is (N+1)/2.

For the parallel form, all characteristics are either single or double pole. Single-pole characteristics require a two-term summation, so two characteristics may be computed in a single P5 element. Double-pole characteristics require a three-term summation, which subsumes a full P6 element. Thus one element implements two poles, whether single or double, and N/2 elements are required to calculate the characteristics portion of the transfer function. The numerator calculations involve a two-term summation for single poles and a three-term summation for double poles, thus requiring a maximum of N/2 elements per pole. The total element count is N.

For the orthogonal polynomial form, a full element (P5) is required for each pole, or for each "rung" of the ladder. The numerator calculation involves an (N+1) term series, so (N+1)/4 P6 elements are needed. Thus the total element count is N + (N+1)/4.

Number of Layers Required for Each Form -- Both the direct and parallel forms may be implemented in two-layered networks of elements, but the orthogonal polynomial form requires an (N+1)-layered network. For the characteristics computation of the

direct and parallel forms, the output of each element does not depend upon the output, within a given iteration of the filter, from any other element. Thus all characteristic calculations may be done in parallel. With the orthogonal polynomial form, however, the input to each rung of the ladder includes the output of the prior rung. Thus the rungs must be computed in series.

(Dependent paths were shown in dotted lines in Figures 52, 53 and 54.) In all three forms, the numerator calculation is dependent upon the characteristic calculations, so one additional layer is necessary.

The minimum number of layers L required for each form is thus:

Direct Form:	$L_d = 2$
Parallel Form:	$L_p = 2$
Orthogonal Polynomial Form:	$L_o = N + 1$

The maximum speed of computation, given a fully parallel network, is limited by L times the speed of one element computation.

9.3 TRAINABLE NONLINEAR FILTERS

9.3.1 CONCEPT

In conventional filter design there are generally two underlying assumptions. First, it is assumed that the precise nature of the operation to be performed on a signal is known. Second, the signal conditioning is generally a linear operation. (There are certain notable exceptions such as waveform hard-limiting in FM demodulation; but even in this example, it is the zero-crossing timing which is of interest -- no information is being extracted from the signal in the limiting process.) If the nature of the transfer function is known and linear, there are many straightforward approaches to designing appropriate filters.

However, in certain interesting classes of signal processing applications, the description of the desired filter may not be known, or if it is known, it may be nonlinear and exhibit instability when implemented directly. If there are empirical data which represent the class of signals to be operated on in such cases, adaptive learning network (ALN) procedures may be used to synthesize filter forms which estimate or predict the desired signal parameter. (See the Appendices for discussion of general ALN methods.) The filters are transversal and are therefore always stable. Questions of stability associated with recursive filters are avoided.

The remaining sections illustrate a procedure for using ALN training techniques to synthesize a nonlinear filter from empirical data. The example used involves estimating the energy within a specified band of a broad-band signal. Though the design of a conventional energy estimator is straightforward, this knowledge was not used in the ALN training process -- an important consideration. An after-the-fact comparison is made, however, between the ALN and conventional systems.

9.3.2 DATA BASE SYNTHESIS

The data base used for designing and for subsequent testing of the ALN was synthesized in the following manner. The ALN was trained to mimic the characteristics of an eight-pole Butterworth filter whose 3db cutoff frequencies, ω_l and ω_h, were equal to 1 and 2, respectively. A normalized frequency range of 0 to 10 was used. (So, for example, a 30 kHz signal could be mapped into this range by interpreting it as divided by 3 kHz.) The frequency response of an ideal eight-pole Butterworth continuous filter is shown in Figure 55).

To form the data base, three sets of 130 time signals, $x_j(t)$, $j = 1, \ldots, 130$, were generated. The need for three sets is due to the ALN synthesis and testing procedures. Each signal (of each set) was constructed as a weighted sum of five sinusoids:

$$x_j(t) = \sum_{i=1}^{I} A_{ji} \sin(\omega_{ji} t + \phi_{ji}) + K_j \qquad (9.8)$$

where the definitions and ranges of the parameters in this equation are as follows:

K_{ji} = Bias Term:	Uniform Random	:	-0.5 to +0.5
A_{ji} = Amplitude:	Uniform Random	:	0 to 1
ϕ_{ji} = Phase :	Uniform Random	:	0 to 2
ω_{ji} = Frequency:	Uniform Random on Log Scale	:	0.2 to 10

Uniform random variables were chosen on a log ω scale so that $\omega = \log^{-1}\omega$ would have more values at the low end of the 0 to 10 range and, hence, near the bandpass region.

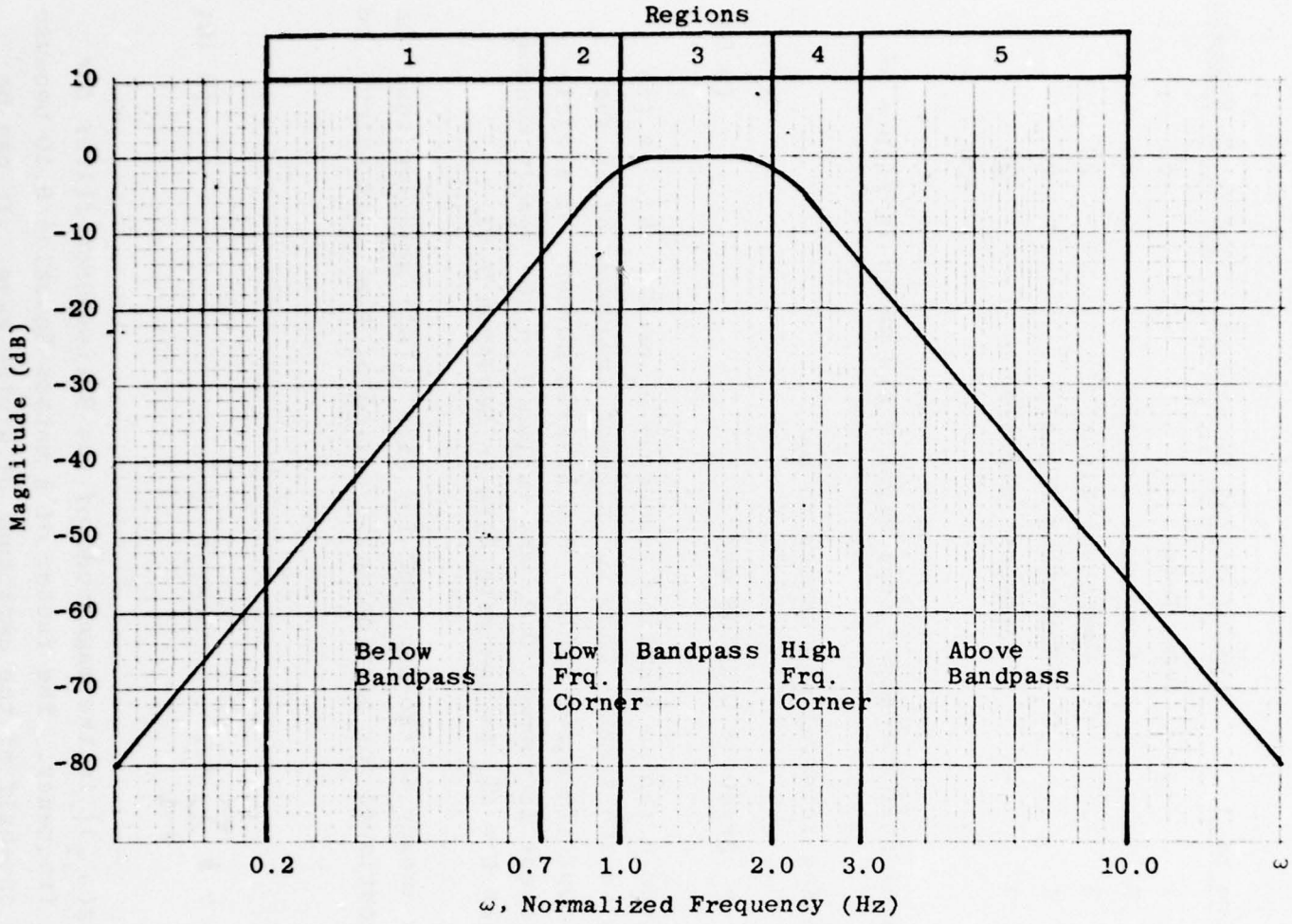


FIGURE 55: MAGNITUDE VERSUS NORMALIZED FREQUENCY FOR AN IDEAL EIGHT-POLE BUTTERWORTH BANDPASS FILTER

The 130 signals (per set) were generated from Equation 9.8 with I given as:

Value of I	Type of Signal Generated	Number of Signals
1	Pure Tone	50
2	2 Component Signals	20
3	3 Component Signals	20
4	4 Component Signals	20
5	5 Component Signals	20
		130

Examples of pure and mixed tone signals are shown in Figure 56.

The sampling interval was chosen to be:

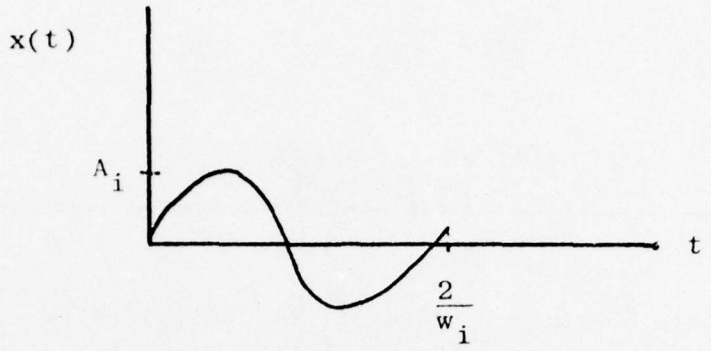
$$T_s = 2\pi/40 = 0.15708 \text{ sec.} \tag{9.9}$$

which gives a radian sampling frequency ω_s of 40. Each $x_j(t)$ was sampled 200 times. Thus the highest frequency of 10 was sampled four times per cycle. One hundred percent of a cycle of the lowest frequency, 0.2, was included in the sample window. (Note that $t = iT_s$ in discrete form in Equation 9.8).

The true energy in the passband for each $x_j(t)$, assuming that the signal continued for an indefinite period of time, was calculated as:

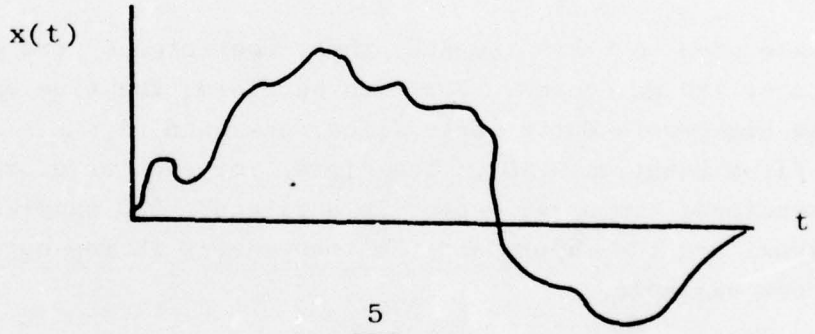
$$e_j = \tfrac{1}{2} \sum_{i=1}^{I} A_{ji}^2 \; | \; g(\omega_{ji}) \; |^2 \tag{9.10}$$

where, $|g(\omega_{ji})|$ is the magnitude of the Butterworth filter for the j_i^{th} frequency. The factor of $\frac{1}{2}$ enters Equation 9.10 because power is one-half of the amplitude of a sine wave. It can be shown that for an eight-pole Butterworth filter the amplitude at frequency ω is obtained from:



$$A_i \sin(w_i t)$$

(a) Single-component



$$x(t) = \sum_{i=1}^{5} A_i \sin(\omega_i t)$$

(b) Multiple-component

FIGURE 56: MULTIPLE HARMONIC INPUT DATA SIGNAL

$$|g(\omega)|^2 = \frac{\left[\omega^2/\omega_1^2\right]^4 \left[\omega_l^2\right]^4 \left[\omega_h^2\right]^4}{\left[(\omega_1 \cos\phi_1)^2 + (\omega + \omega_1 \sin\phi_1)^2\right]\left[(\omega_1 \cos\phi_1)^2 + (\omega - \omega_1 \sin\phi_1)^2\right]}$$
$$\left[(\omega_1 \cos\phi_2)^2 + (\omega + \omega_1 \sin\phi_2)^2\right]\left[(\omega_1 \cos\phi_2)^2 + (\omega - \omega_1 \sin\phi_2)^2\right]$$
$$\left[(\omega_h \cos\phi_1)^2 + (\omega + \omega_h \sin\phi_1)^2\right]\left[(\omega_h \cos\phi_1)^2 + (\omega - \omega_h \sin\phi_1)^2\right]$$
$$\left[(\omega_h \cos\phi_2)^2 + (\omega + \omega_h \sin\phi_2)^2\right]\left[(\omega_h \cos\phi_2)^2 + (\omega - \omega_h \sin\phi_2)^2\right]$$

(9.11)

where $\omega_l = 1$, $\omega_h = 2$,

$\phi_1 = 22.5^o$, $\phi_2 = 67.5^o$.

The data base used to train the ALN, then, consisted of two of the three sets of 130 waveforms. For each waveform, the true energy within the eight-pole Butterworth filter passband ($1 < \omega < 2$) is given (from Equation 9.10). Therefore, for each waveform, a 201-dimensional component vector is obtained: 200 samples of the signal are the inputs and the true energy is the output, or dependent, variable.

9.3.3 DATA BASE PARAMETERIZATION

In order to reduce the number of inputs and, potentially, generate more meaningful inputs, parameters of the 200 samples from each waveform were computed. These parameters were arbitrarily chosen, and there is no reason to believe that they are the best, or even a good set. The definitions of the 31 parameters are given in Table 12. The width of the 10 histogram bins were arbitrarily chosen as: 0 to 0.01, 0.01 to 0.022, 0.022 to 0.047, 0.047 to 0.10, 0.10 to 0.22, 0.22 to 0.47, 0.47 to 1.0, 1.0 to 2.2, 2.2 to 4.7, and 4.7 to ∞.

TABLE 12

WAVEFORM PARAMETERS

Number	Description
1	Mean signal value: $\bar{x}_j = \frac{1}{200} \sum_{m=1}^{200} x_j(m)$
2	Standard deviation of signal mean: $\sigma_{\bar{x}_j} = \left[\frac{1}{199} \sum_{m=1}^{200} (x(m) - \bar{x}_j)^2 \right]^{\frac{1}{2}}$
3	Maximum deviation from signal mean: $d_j = \max_{m=1, \ldots, 200} \lvert x_j(m) - \bar{x}_j \rvert$;
4	Mean magnitude of point-to-point difference: $\bar{d}_j = \frac{1}{199} \sum_{m=2}^{200} \lvert x_j(m) - x_j(m-1) \rvert$
5	Standard deviation of point-to-point differences: $\sigma_{\bar{d}_j} = \left[\frac{1}{198} \sum_{m=2}^{200} (\lvert x_j(m) - x_j(m-1) \rvert - \bar{d}_j)^2 \right]^{\frac{1}{2}}$
6-15	Number of hits in each of 10 bins of the point-to-point differences histogram: $\lvert x_j(m) - x_j(m-1) \rvert$
16-25	Number of hits in each of 10 bins of the signal deviations about mean histogram: $\lvert x_j(m) - \bar{x}_j \rvert$
26-31	Second, third, and fourth moments of the two above histograms.

9.3.4 NETWORK TRAINING

The within-band energies of the 130 signals computed from Equation 9.8 varied over a dynamic range of greater than 12 decades. Since the network training process synthesizes functions which minimize squared errors, and it is desirable in this example problem to predict the small energies (within the passband) very accurately, the network was trained to predict the log of the energy rather than the energy itself:

$$y_j = 10 \log_{10} e_j \tag{9.12}$$

In the training process, the first set of 130 signals was used to synthesize the network. The second set of 130 signals was used to avoid overfitting the polynomial energy estimation model. The third set of 130 signals was used to evaluate the resultant network performance.

A block diagram of the resultant energy estimation system is shown in Figure 57. The input signal s(t) is passed through a preprocessor which extracts the 31 parameters. This information forms an input vector to the ALN, which then estimates the log of the energy. The antilog is computed to yield the estimated energy, e, within the desired passband.

9.3.5 RESULTS

The results of the network training are presented in two forms. First, energy estimates for the signals of mixed frequencies were accurate to within 5 dB on the average over a range of 60 dB. These accuracies are based on 130 mixed frequency signals which were <u>not</u> used in the network training process.

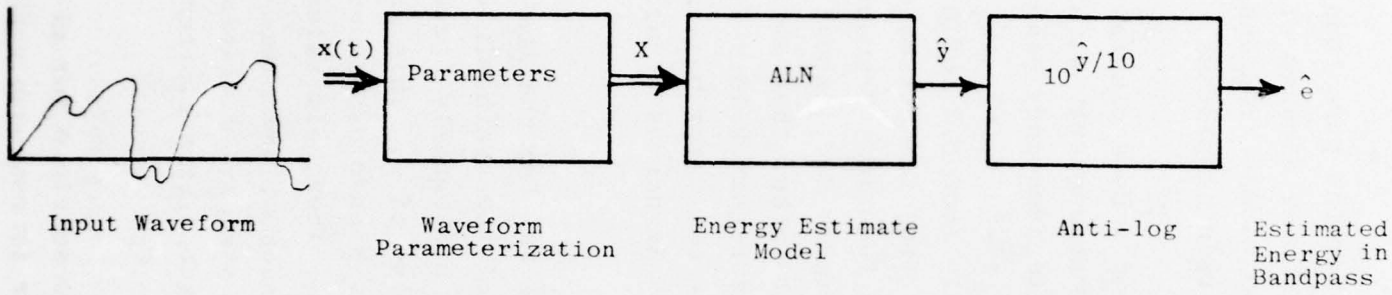


FIGURE 57: ALN ENERGY ESTIMATION SYSTEM

Second, energy estimates for the pure tone signals were estimated within 6 dB on the average over a range of 60 dB. These accuracies are based on 50 single frequency signals which were not used in the network training process.

It is also possible to plot a Bode diagram of the ratio of total signal energy to the estimated energy within the band. This was done by constructing single frequency sinusoids (I = 1 in in Equation 9.18) with $A_{ji}=0.5$, $\phi_{ji}=0$, and $K_j=0$. The frequency term was then varied slowly from 0.2 to 10. The resultant signal was parameterized, and its true and estimated e_j found via Equation 9.11 and the ALN function, respectively. Ideally, the response would resemble Figure 9.7, except that the decible scale would be in terms of power rather than amplitude. The plot for the ALN system is shown in Figure 9.10. It can be seen in this figure that the ALN energy estimator nas in fact learned the overall trend, but that it is not as accurate as desired in the passband region.

The pair of curves (Figure 58) for the ALN system is a measure of the phase sensitivity of the synthesized function. This was estimated by choosing six equally-spaced values of ϕ in the range 0 to 2π for each value of ω. This gave, for each value of ω, six single-component sinusoids possessing the same frequency but with different phases. Then, six values of the estimated energy were obtained for each ω, and these were averaged. The mean energy plus and minus its standard deviation are actually plotted in Figure 58, so that, for each ω, the plotted upper value is $e + \sigma_e$ and the lower value is $e - \sigma_e$.

Since these two values are so close for all ω, it was concluded that the trained estimator is reasonably phase insensitive.

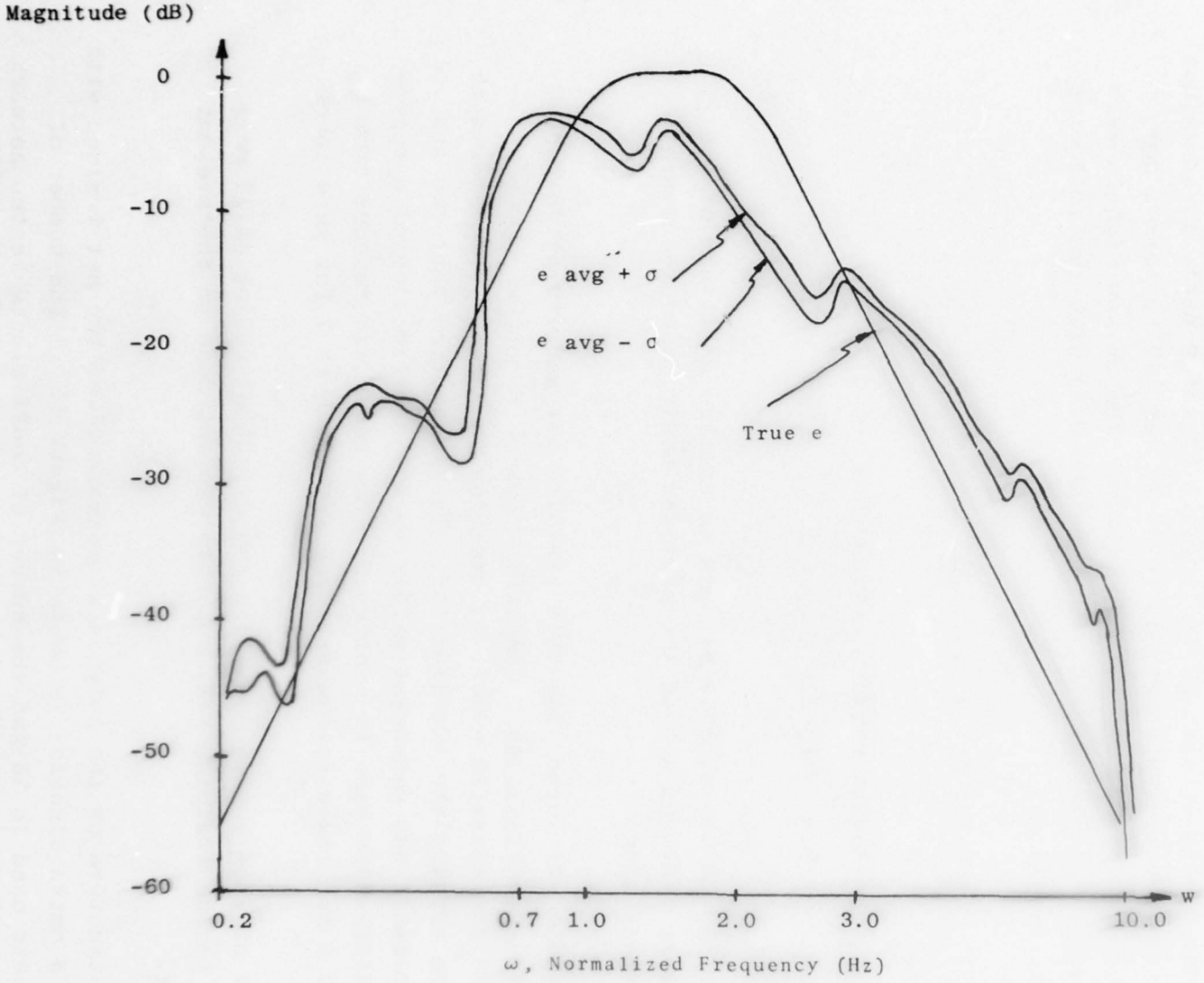


FIGURE 58: ENERGY BAND PASS FREQUENCY RESPONSE FOR THE ALN SYSTEM

Upon closer examination of the 130 signals used to train the polynomial function, it was found that less than 15 percent of the 130 signals had any appreciable amplitude in the passband region. So, on the assumption that the performance (Figure 9.10) was due mainly to lack of appropriate training data, an additional 36 signals were generated and added to the 130 to form a new, augmented training set of 166 signals. The 36 new signals were all single sinusoids (I = 1 in Equation 9.8) with the following characteristics:

K_j = 0 for j = 1, ..., 36

ϕ_j = 0

A_j = Random uniform between 0.3 and 0.7

ω_j = 0.5 + (j-1)0.1: j = 1, ..., 36

Thus, the 36 new signals all had an average amplitude of 0.5 and were centered around the passband position of the overall frequency range.

A new ALN was found from this training set and its performance is given in Figure 59. Comparing Figures 58 and 59 illustrates the dramatic effect of creating the energy estimater with a more appropriate training set. The assumption that the ALN performance was dependent mainly on the availability of a representative data base is confirmed. The error was reduced from 5 dB to 4 for mixed frequencies and from 6 dB to 3 for pure tones.

It is of interest to establish the effect of adding still more data to the training set, but this was not done in the present study.

The structure of the polynomial network (i.e. the net trained with the 36 extra signals) is shown in Figure 60. The number of elements used is 26 and the number of coefficients in the network is 146.

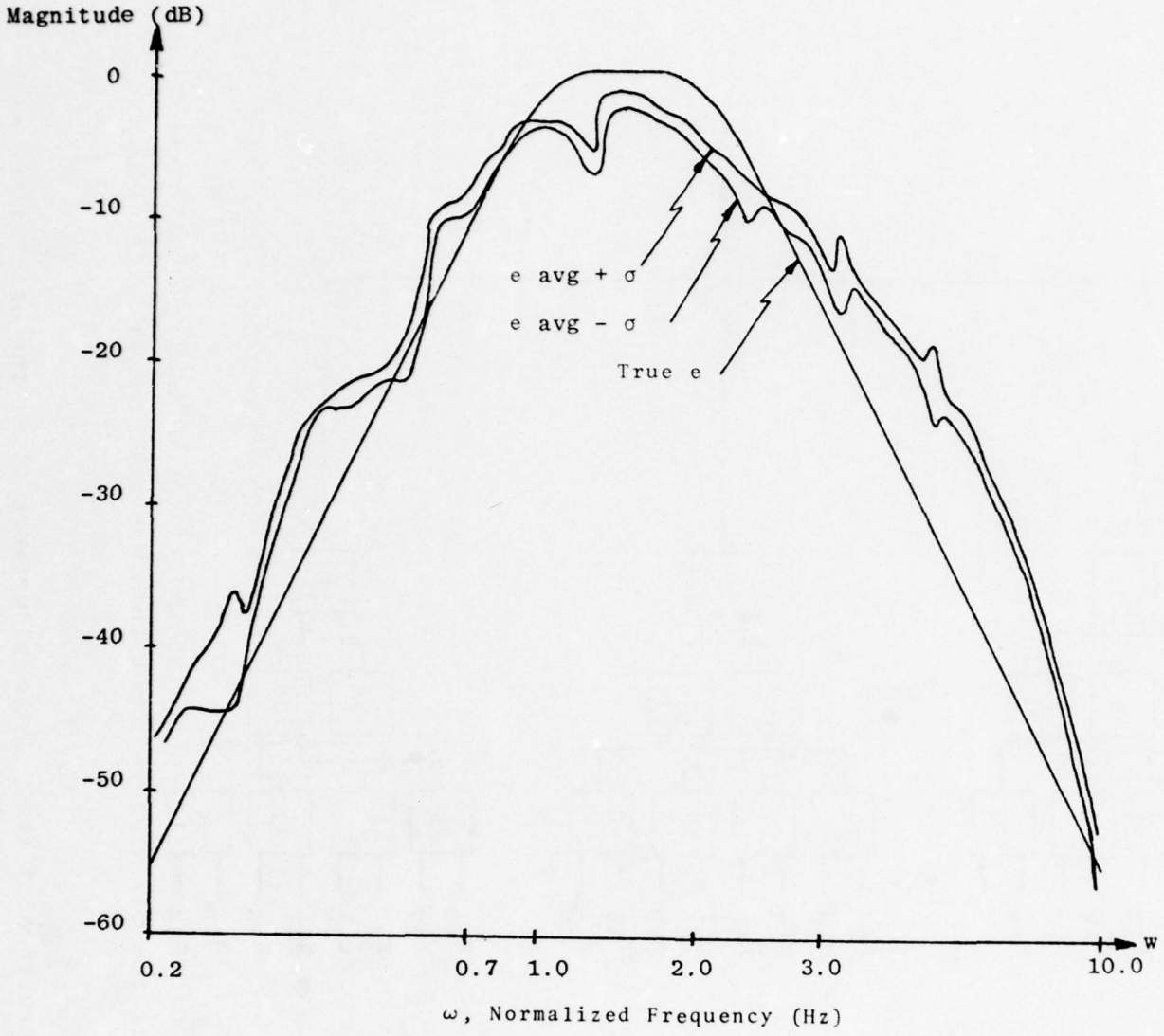


FIGURE 59: ENERGY BAND PASS FREQUENCY RESPONSE FOR THE NEW ALN SYSTEM

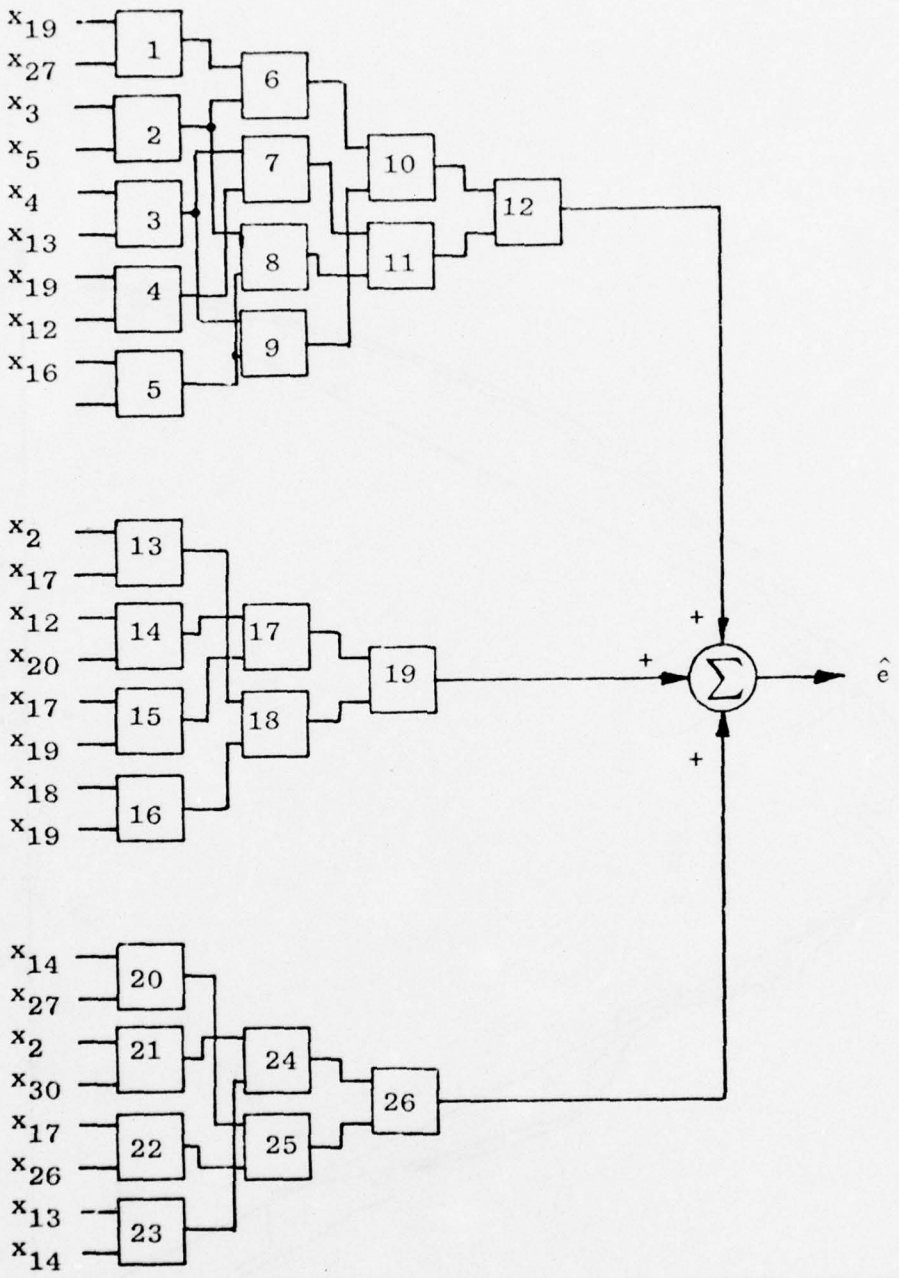


FIGURE 60: ALN FOR BAND PASS ENERGY PREDICTION SYSTEM

A block diagram of a conventional energy band estimator is shown in Figure 61 for comparative purposes to Figure 60.

9.3.6 DISCUSSION

Though the conventional system is a more accurate estimator of desired waveform property -- particularly within the passband -- it was designed with a priori knowledge of the formula for computing the energy parameter. The network system was designed by a fully automated process which had no a priori knowledge of any formula for the parameter. The network training algorithm synthesized its own formula from empirical data. The mathematical equations of the network system appear to be significantly different from those of the conventional system, but the results are comparable.

The effect of the ALN filtering action can be viewed in the time domain in the following way. A signal to be filtered can be generated from Equation 9.8 as:

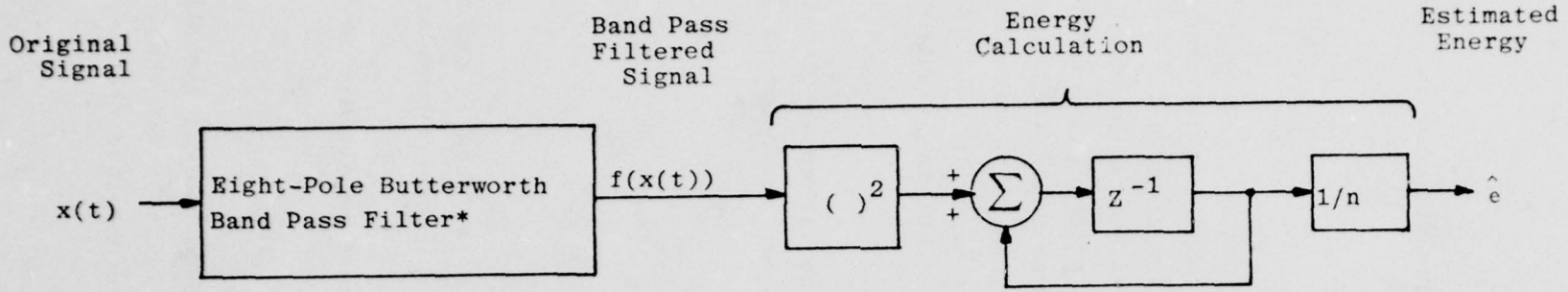
$$x(t) = \sum_{i=1}^{I} A_i \sin(\omega_i t + \phi_i) + K_i$$

The amplitude factors, A_i, should properly be written in the time domain as $A_i(t)$. In the frequency domain, these magnitudes are $A_i(\omega)$.

The effect of the ALN filter is to alter these magnitudes as:

$$A_i'(\omega) = A_i(\omega) \mid H(\omega) \mid$$

where, $\mid H(\omega) \mid$ is the magnitude from the Bode plot of the ALN filter. Ideally, $H(\omega)$ would resemble Figure 55 for the ω of interest. In actuality, these values are obtained from Figure 59.



*Digital linear filter of standard direct, parallel, or orthogonal polynomial form, containing 8 stages.

FIGURE 61: CONVENTIONAL ENERGY ESTIMATION SYSTEM

In order to reconstruct the signal, x(t), and to observe the filtering effect, the time domain-representation of $A_i'(\omega)$ are found and:

$$x'(t) = \sum_{i=1}^{I} A_i' \sin(\omega_i t + \phi_i) + K_i$$

One comparison between x(t) and x'(t) would be their cross correlation.

9.4 CONCLUSIONS

The ALN training procedure is a very powerful tool for synthesizing a nonlinear filter to estimate a waveform parameter whose precise mathematical definition is not known *a priori* but is inherently represented within empirical data.

ALN systems have the advantage that they fall within the transversal filter category and thus do not have round-off accuracy and numerical stability problems often associated with recursive filters and particularly nonlinear recursive systems.

Another advantage of the ALN system is that it is *trainable* via a flexible software/hardware configuration. This means that the same hardware can be *configured* to extract signal parameters and to solve problems that arise from time to time. Therefore, their usefulness is not limited to preprogrammed and structured algorithms.

SECTION X

NON-POLYNOMIAL PREPROCESSOR FUNCTIONS

10.1 INTRODUCTION

The present element circuitry developed by RCA under Contract F33615-73-C-1089 is capable of computing the following mathematical functions:

P1: $y = a_0 + a_1x_1$

P2: $y = a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2$

P3: $y = a_0 + a_1x_1 + a_2x_2 + a_3x_1x_2 + a_4x_1^2 + a_5x_2^2$

These functions may be used as building blocks for many-variable, high degree polynomials and are thus suitable for the computation of polynomial networks and for many preprocessing functions. However, there are several types of preprocessing functions which are typically used in waveform processing applications where polynomial networks are used but which cannot be computed within the present element design.

It is desirable to augment the computing capabilities of the present element so that all preprocessing and network computations may be performed in the same piece of hardware. The following functions are often used in signal processing applications:

(a) Counting the number of zero crossings in a time series.
(b) Counting the number of threshold crossings in a time series.
(c) Finding the peak value of a time series.
(d) Rectification.
(e) Hard limiting.
(f) Sign detection.
(g) Variable threshold detection.
(h) Dual threshold detection.

This section presents preliminary functional designs, circuit diagrams, and timing diagrams for the above functions. All functions have been combined into a single computational unit to minimize total circuitry. The functions include processing on two's complement, 24-bit fixed point words.

The objective of the work is to provide a baseline design for a nonpolynomial processor which would ultimately undergo large scale integration for use along with the polynomial processor. The circuitry contains approximately 3,100 equivalent gates. It is estimated that approximately 2,000 additional gates would be required to expand the unit with exponent logic to provide a full floating point capability.

10.2 FUNCTIONAL DESCRIPTION

Since the polynomial element is designed to receive eight data inputs simultaneously, the nonpolynomial logic is configured the same way to provide maximum compatibility. The eight inputs are word parallel, bit serial. The eight input words are stored in a set of eight twenty-five bit registers. The twenty-fifth bit, at the leading edge of the word, is a repetition of the sign bit and is used to prevent overflow when adding two's complement twenty-four bit words.

The typical computation cycle for the nonpolynomial logic consists of a data input subcycle where the eight words are clocked in in parallel with a series of 25 clocks, and an output subcycle where the selected function is simultaneously computed and clocked out with a series of 25 clocks. The computational form which implements these two subcycles is shown in Fig. 62.

In general, each of the functions of the nonpolynomial processor requires several full cycles to process a waveform. With one exception (rectification), the first cycle on any time series analysis is an initialization cycle and consists of loading the analysis parameters, e.g., thresholds or limiting values. The subsequent cycles involve the loading and processing of the time series data.

There are five registers for the storage of analysis parameters. Some functions, for example the zero crossing counter, require that cumulative results be maintained over several "cycles" of processing. There are nine registers for intermediate results storage. Thus, there are a total of twenty-two data storage registers -- eight for time series data, five for analysis parameters, and nine for intermediate results.

For consistency of operation, a common set of clocks is used for all functions and is used on both initialization and processing cycles.

10.3 LOGIC DESCRIPTION

The logic diagrams for the nonpolynomial processor are shown in Figures 64 through 70 (at the end of this section). Descriptions of the logic for the individual functions are presented in this section.

A timing diagram for the clocks is shown in Fig. 63. Definitions for each of the clocks are as follows:

- C_{in} - 24 pulse train to clock data into the processor. (The input data are stored in the data input registers on Figure 3.)
- $C_{sb_{in}}$ - Single pulse on input subcycle which clocks the sign bit of the incoming data into sign bit storage flip-flops (see Figure 4). Occurs simultaneously with 24^{th} C_{in} pulse.
- C_{in}' - The 25^{th} clock on the input subcycle; Repeats the sign bit; Occurs immediately after the 24^{th} C_{in} pulse.
- C_{in}'' - Occurs immediately after C_{in}'. Is a single pulse. (Not used).
- C_{out} - 24 pulse train to clock out result of nonpolynomial processor. Occurs after C_{in}''
- $C_{sb_{out}}$ - Clocks the sign bit into intermediate storage. Occurs simultaneously with 24^{th} C_{out} pulse.
- Clear - Clears logic for next cycle. Occurs immediately prior to C_{in}.

The eight functions specified in Section 1 may be implemented with five logic groups. Counting the number of zero crossings is done with the threshold crossing count logic by setting the threshold to zero. Sign detection and threshold detection are performed with the dual threshold detection by setting both thresholds to the same value to obtain threshold detection, and by setting both the thresholds to zero to obtain sign detection. The five different functions are specified by a three-bit control word as follows:

Function Select Lines	Function	Schematic Designations
000	Count threshold crossings	FTC
001	Find peak value	FYMAX
010	Rectification	FRECT
011	Hard limiting	FHL
100	Dual threshold detection	FDTD

The logic controlling the function select is shown in Figure 70.

10.3.1 COUNTING THE NUMBER OF THRESHOLD CROSSINGS IN A TIME SERIES

Two types of input data are required to compute the number of threshold crossings in a time series: the threshold value and the time series samples. The threshold is input the processor on the initial cycle, and the series data is input sequentially, in groups of eight, on the subsequent cycles.

The initialize cycle is so noted by setting the initialize line high during the input subcycle. The threshold is input on the first of the eight data input lines and the initialize line will cause it to be stored in the first parameter storage register shown in Figure 64. The cumulative crossing register shown on the lower right portion of Figure 66 is also set to zero on the initialize cycle.

On the second cycle, the first eight time samples are input to the eight data lines. As shown in Figure 65 (left side), the threshold is subtracted from each of the eight data words. This computation occurs on the input subcycle. The signs of the resultant subtractions, which indicate whether each signal is above or below the threshold, are stored in a set of eight flip-flops.

As shown in Figure 66, threshold crossings are detected by a series of exclusive-or gates which detect, between one data point and the next, a change in the sign bit. If the sign bit between two successive data points does change, it indicates that the signal crossed the threshold between those two points and the crossing count should be incremented.

Four three-bit adders are used to count up to seven crossings between the eight data points. This sum is added to the cumulative threshold crossing register on the output subcycle. The new cumulative is passed to the processor output logic for use when the time series is complete and is also passed back into the cumulative register for use on the next cycle if the series is not complete. Intermediate sums may be read out if so desired by the host processor.

Note that although time series data points 1 through 8 were input on the first data cycle, points 8 through 15 must be input on the subsequent cycle. The eighth point from one cycle must be repeated as the first on the next cycle so that a crossing between that point and the next one in the time series may be detected.

10.3.2 FINDING THE PEAK VALUE OF A TIME SERIES

Like the threshold crossing counter, the peak value detector operates on sequential portions of the time series. However, the data sequence may be modified. For the peak value detector, the first eight samples, x_1 through x_8, are input to the nonpolynomial logic on the first cycle, x_9 through x_{16} on the second cycle and so forth.

The peak value register, at the right side of Figure 66 is set to the smallest possible negative number (-2^{23}) on the initialize cycle.

The peak value function operates in a pipe-line decision-tree fashion where there are four levels of decision making. In the first level, four pairs of data are compared to find four pairwise maxima. In the second level, two four-way maxima are found, and in the third level, the eight-way maximum is found. The fourth decision level involves the comparison of the eight-way maximum and the prior time series maximum (peak value, x_{max}, register).

Though there are four decision levels, the decision functions overlap in computation cycles. While the eight input data words are being input to the eight buffer registers, Figure 64, they are simultaneously input to four pairwise comparitors, Figure 66, and being compared. On the computation cycle (the comparison having actually been made on the input subcycle), the higher of each pairwise data is selected for transmission to the second decision level.

The second level is designed the same as the first; however, the input cycle to the second level corresponds (in time) to the output cycle of the first level. Similarly, the third level input cycle corresponds to the second level output cycle, and the fourth level input cycle corresponds to the third level output cycle.

The net delay from the eight data word input to the fourth level decision is two and one-half cycles. Thus after the original time series is complete, two additional input-output cycles must be exercised before the final output result has been processed through the pipe-line.

10.3.3 RECTIFICATION

The rectification function simply inverts negative numbers to make them positive. Eight data words are processed in parallel on each cycle. No initialization cycle is required.

On the input subcycle, the sign bits of the data are stored in the flip-flops shown on Figure 65. The data outputs are controlled by these sign bits. If the sign is positive (0), the data is clocked out from the input register as it was input. If the sign was negative (1), all the bits are inverted and serially added to 00...01, to give the rectified value.

10.3.4 HARD LIMITING

The hard limiting function takes a time series input and produces a time series output which is equal to the input unless the input value exceeds an upper threshold, TU, or is less than a lower threshold, TL.

On the initialization cycle, the limiting values are placed in corresponding parameter storage registers, Figure 62. TU and TL are gated from the first and second data inputs respectively.

Successive cycles produce the hard limited values from the time series data. The time series data are stored in the data input registers, Figure 62, and are simultaneously subtracted from the upper and lower limits, as shown in Figure 65. The sign bits of the resultant subtractions are clocked into flip-flops, Figure 65, and are used to control the data output, Figure 68. If x_i is greater than the upper limit, the sign of x_i-TU will be positive (sign bit equal to 0), and the upper limit value TU will be clocked out as y_i. If x_i is less than the lower limit, the output y_i will be equal to the lower limit TL. If neither of those conditions exist, x_i is between the limits, and y_i will be equal to x_i.

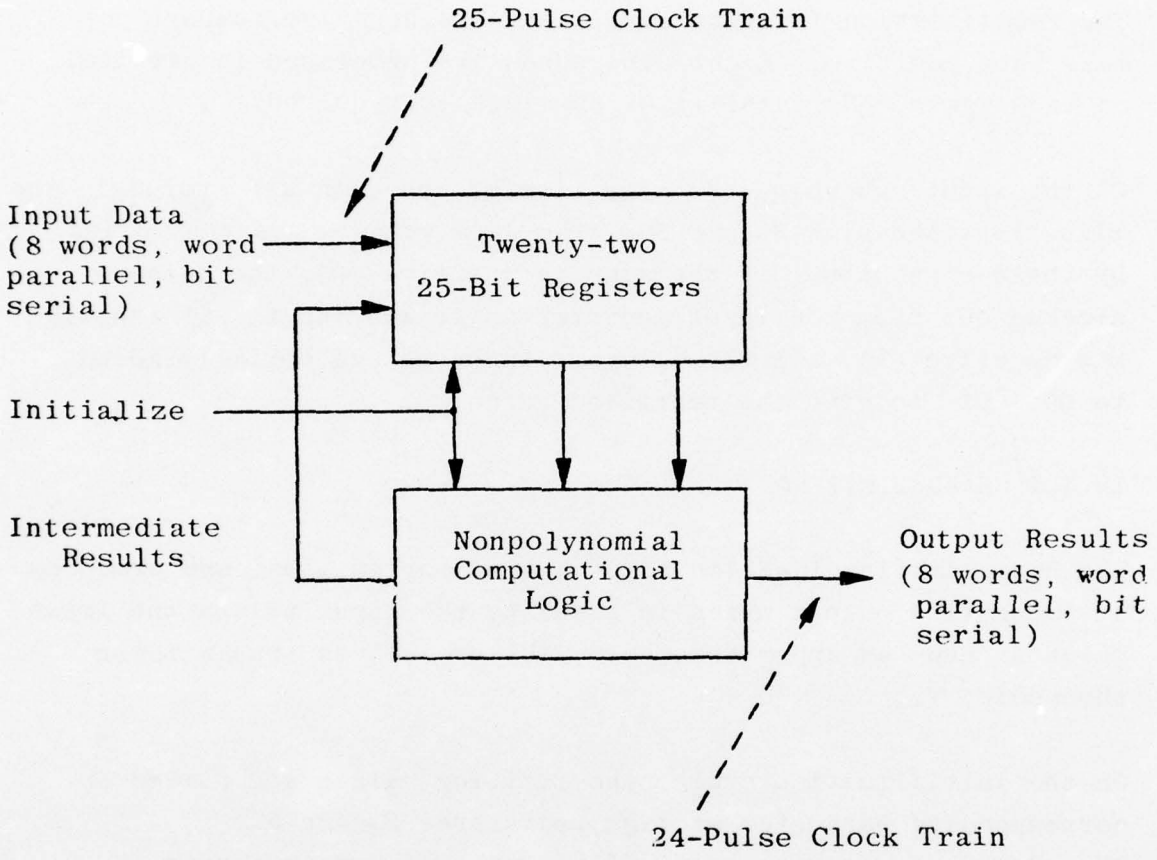


FIGURE 62 COMPUTATIONAL FORM OF NONPOLYNOMIAL PROCESSOR

10.3.5 DUAL THRESHOLD DETECTION

In dual threshold detection, the output y_i is A, B, or C, depending upon whether the input x_i is above an upper threshold, between the upper and lower thresholds, or below the lower threshold. The values of A, B, C, (typically (+1, 0, -1) are user specified on the initialization cycle, along with the thresholds TU and TL. TU and TL are put in via the first two data inputs, respectively, and A, B, and C are put in through inputs 3, 4, and 5.

The time series data are input in groups of eight on successive cycles. As shown in Figure 65, TU and TL are simultaneously subtracted from each of the data words, and the resulting sign bits indicate whether the data is above the threshold (sign bit positive, 0) or below (sign bit negative, 1). The subtraction and sign bit storage are accomplished on the input subcycle.

The values A, B, or C are clocked out on the output cycle according to the resulting subtraction sign bits. If x_i is greater than TU, A is clocked out. If x_i is less than TL, C is clocked out. If neither of those conditions exist, x_i is between the thresholds and B is clocked out.

10.3.6 SIMULTANEOUS OPERATION OF MULTIPLE FUNCTIONS

The way the logic is configured, it is possible to perform some of the functions simultaneously. The data from a time series may be processed once and the following combination of functions may be computed in parallel:

- Count of threshold crossings
- Find the peak value
- One of following three:
  - Rectification
  - Hard limiting
  - Dual thresholding

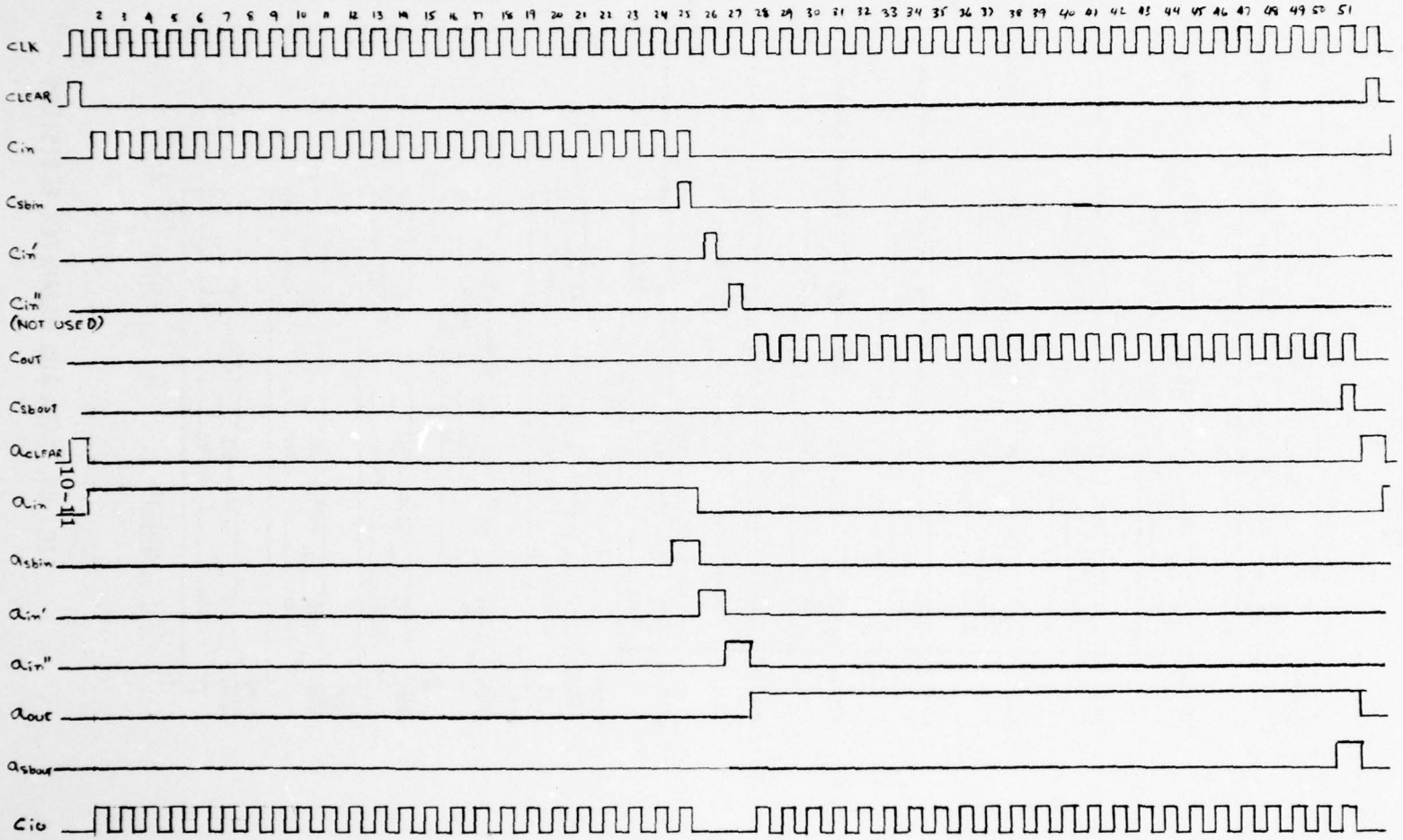
This simultaneous processing is possible because separate logic is used to compute the different functions, and because the threshold-crossing-count and peak-value data need not be output until after the whole time waveform has been processed.

As the waveform is being processed, the rectified, limited, or thresholded time waveform may be output by appropriate control of the function select lines. After the waveform has been processed, the function select lines may be modified on the subsequent cycle to obtain the threshold crossings. On the third cycle after the processing, the waveform peak value will be available.

It must be noted, however, when performing multiple functions simultaneously, that the input data sequence be properly handled. In the above example, the data sequence is dictated by the threshold crossing protocol, where the last data point on one cycle is duplicated on the first point on the next. This will not impact the results of the peak detection logic, but care must be taken in handling the output waveform because of the data duplication.

Significant time savings may be achieved by simultaneous computation of the functions.

FIG. 63 CLOCK PULSE TIMING DIAGRAM



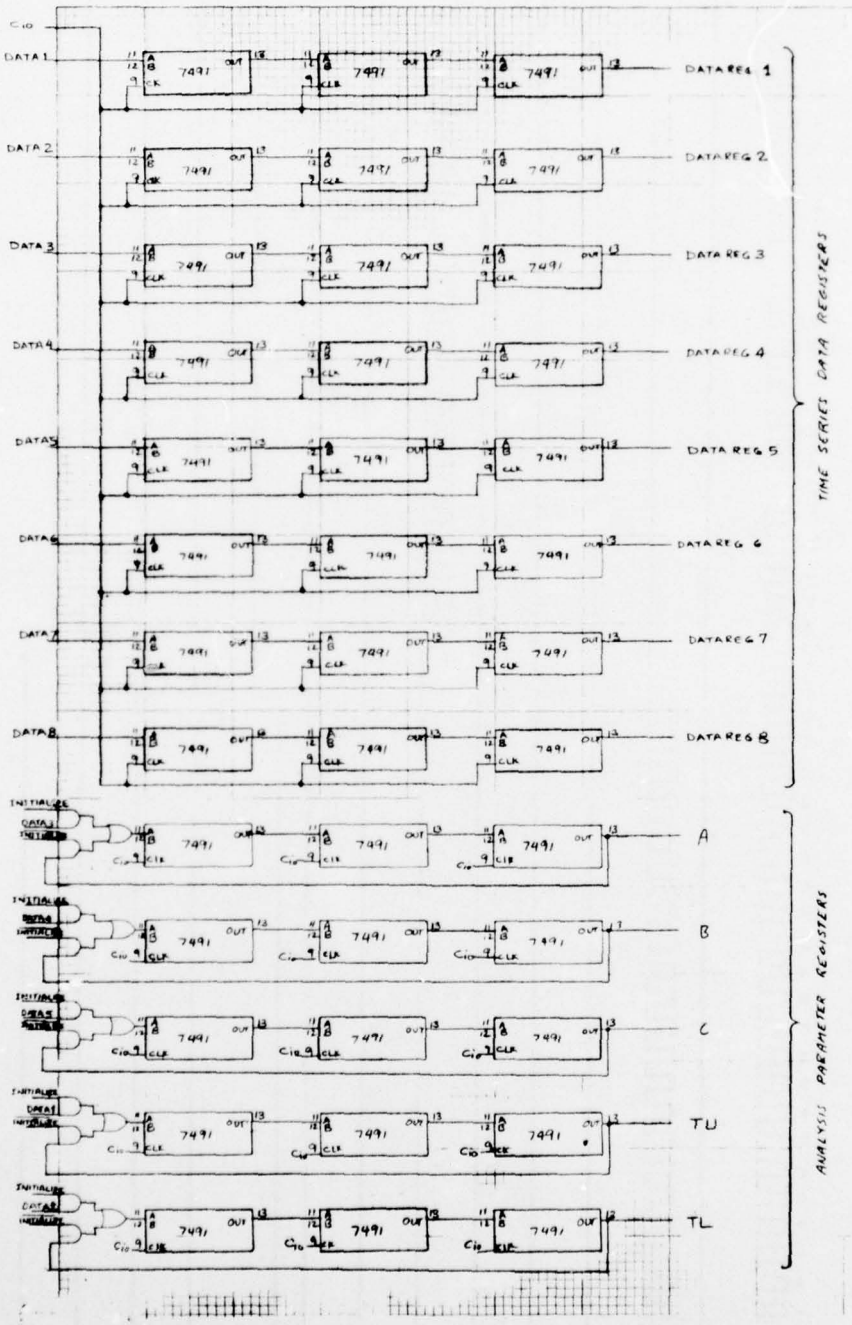


FIG. 64 DATA AND PARAMETER REGISTERS

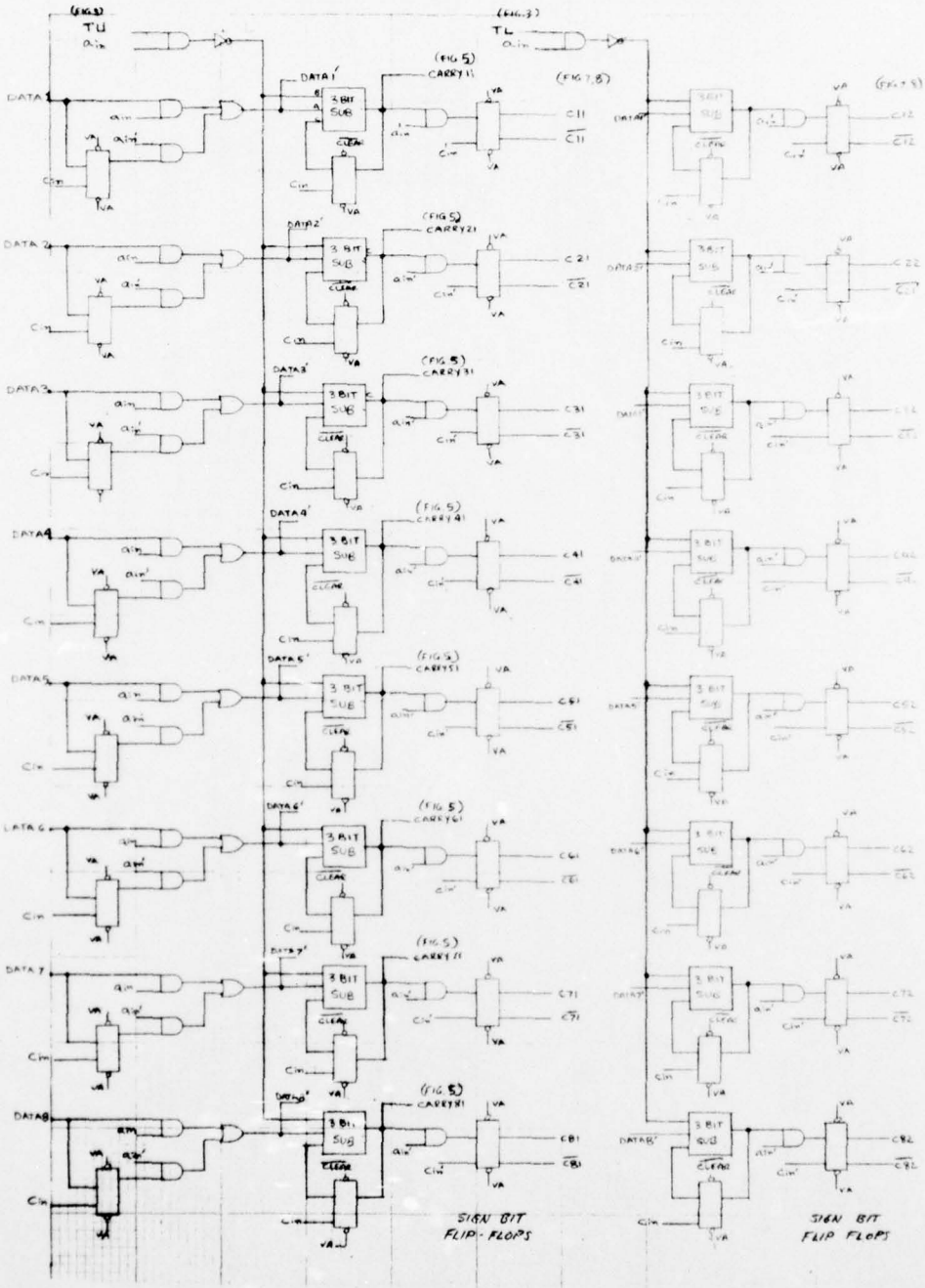


FIG. 65 CALCULATIONS OF SIGN OF X MINUS THRESHOLD

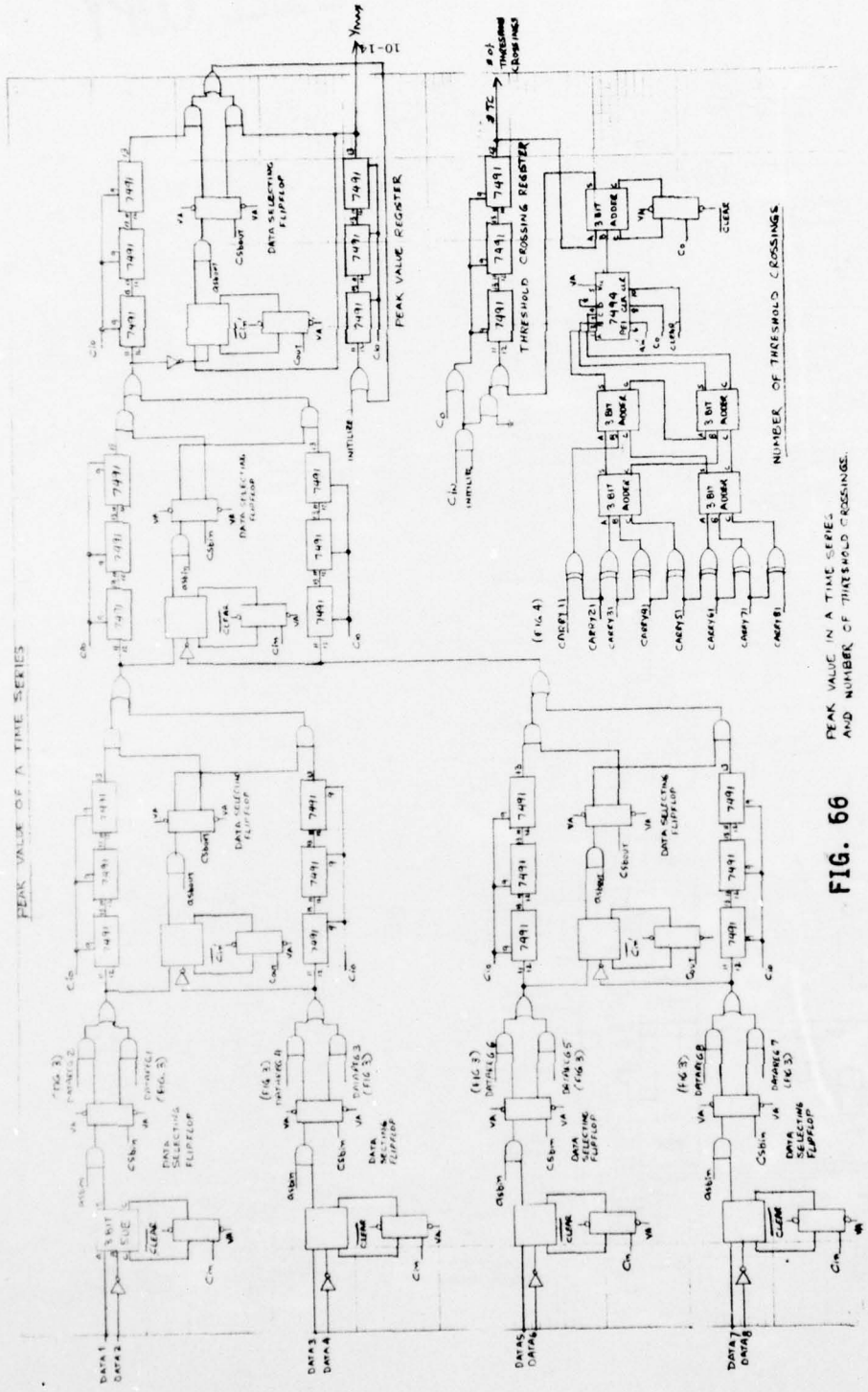


FIG. 66 PEAK VALUE IN A TIME SERIES AND NUMBER OF THRESHOLD CROSSINGS.

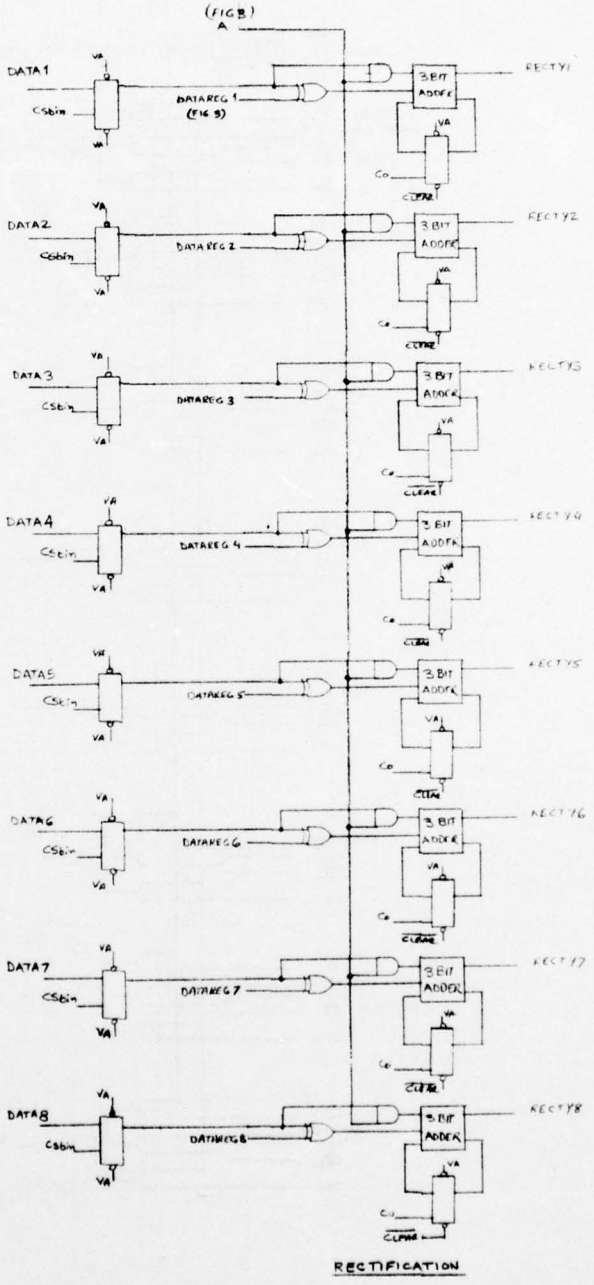


FIG. 67 RECTIFICATION

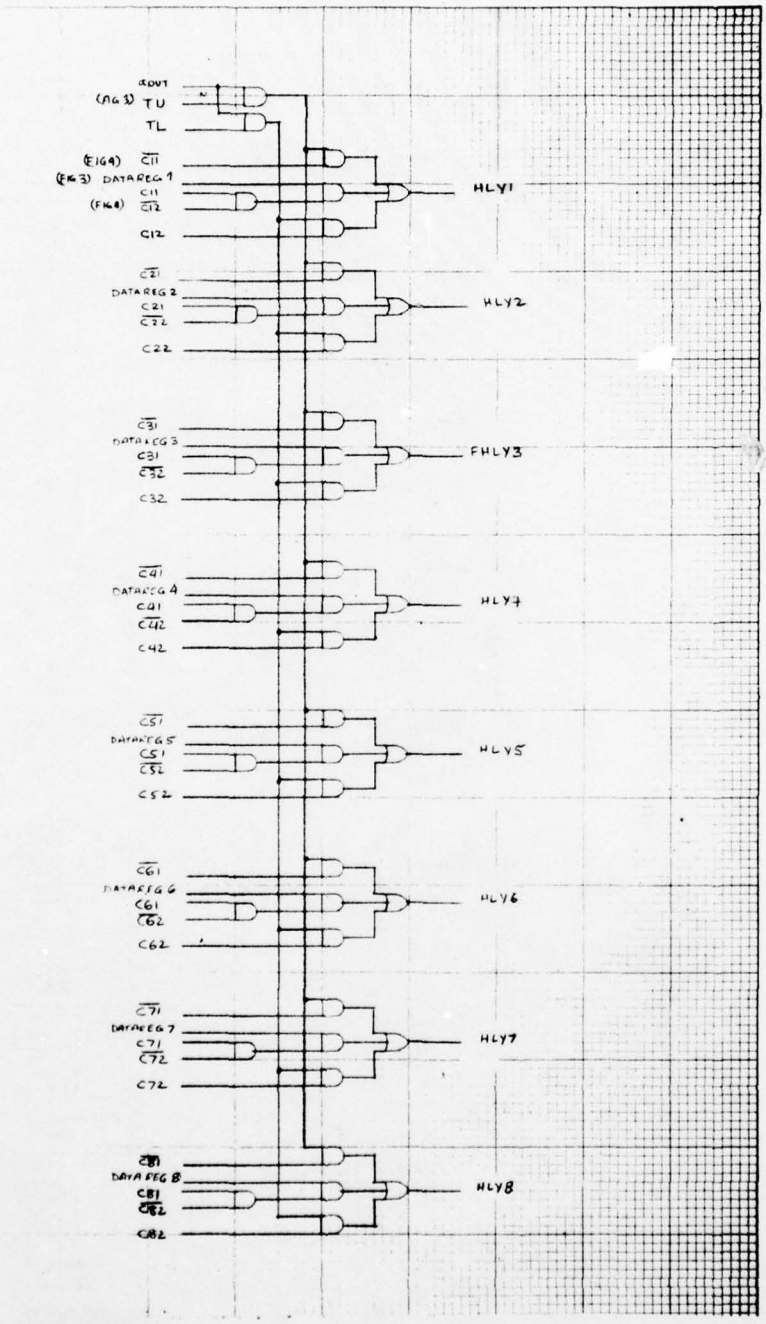


FIG. 68 HARD LIMITING

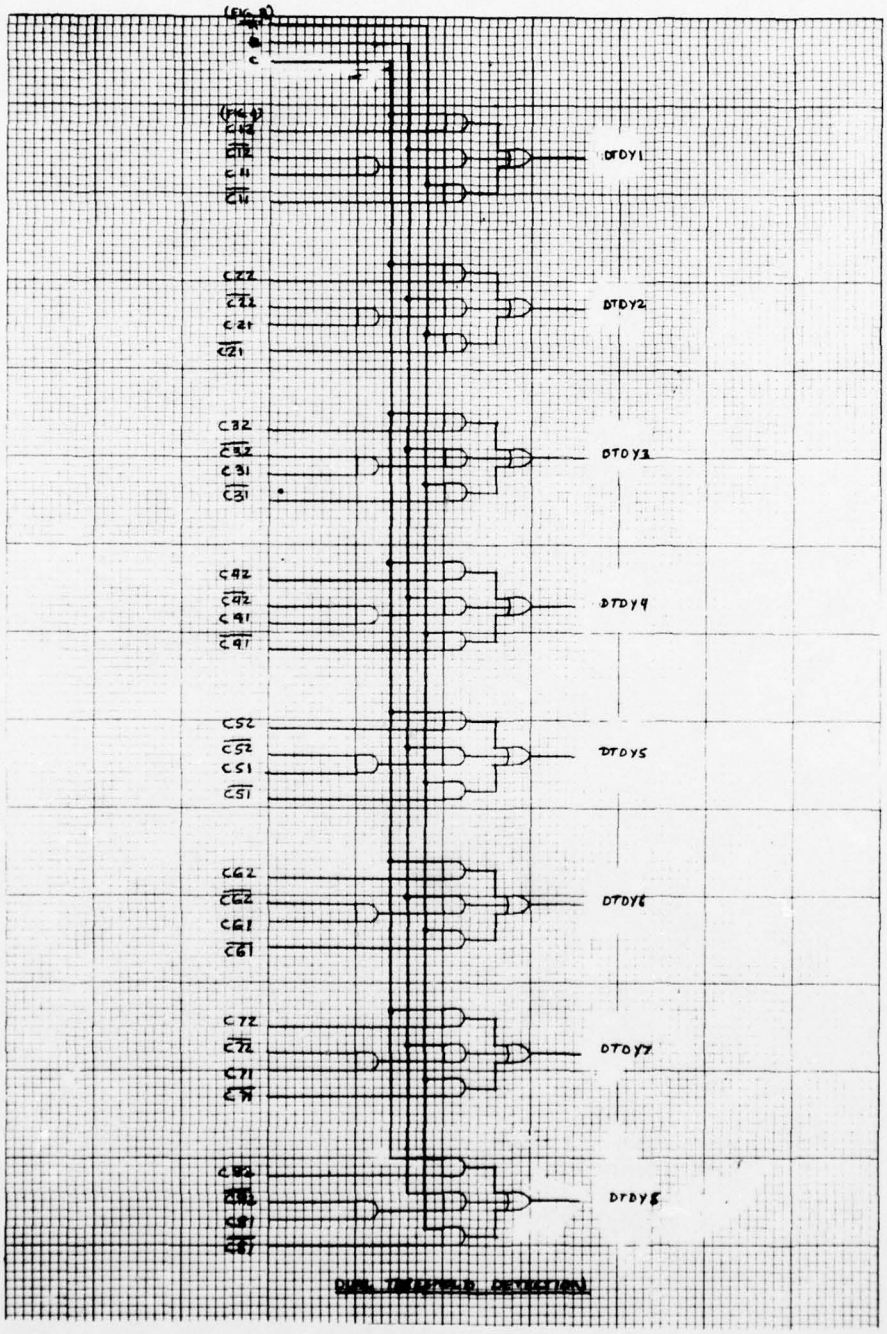


FIG. 69 DUAL THRESHOLD DETECTION

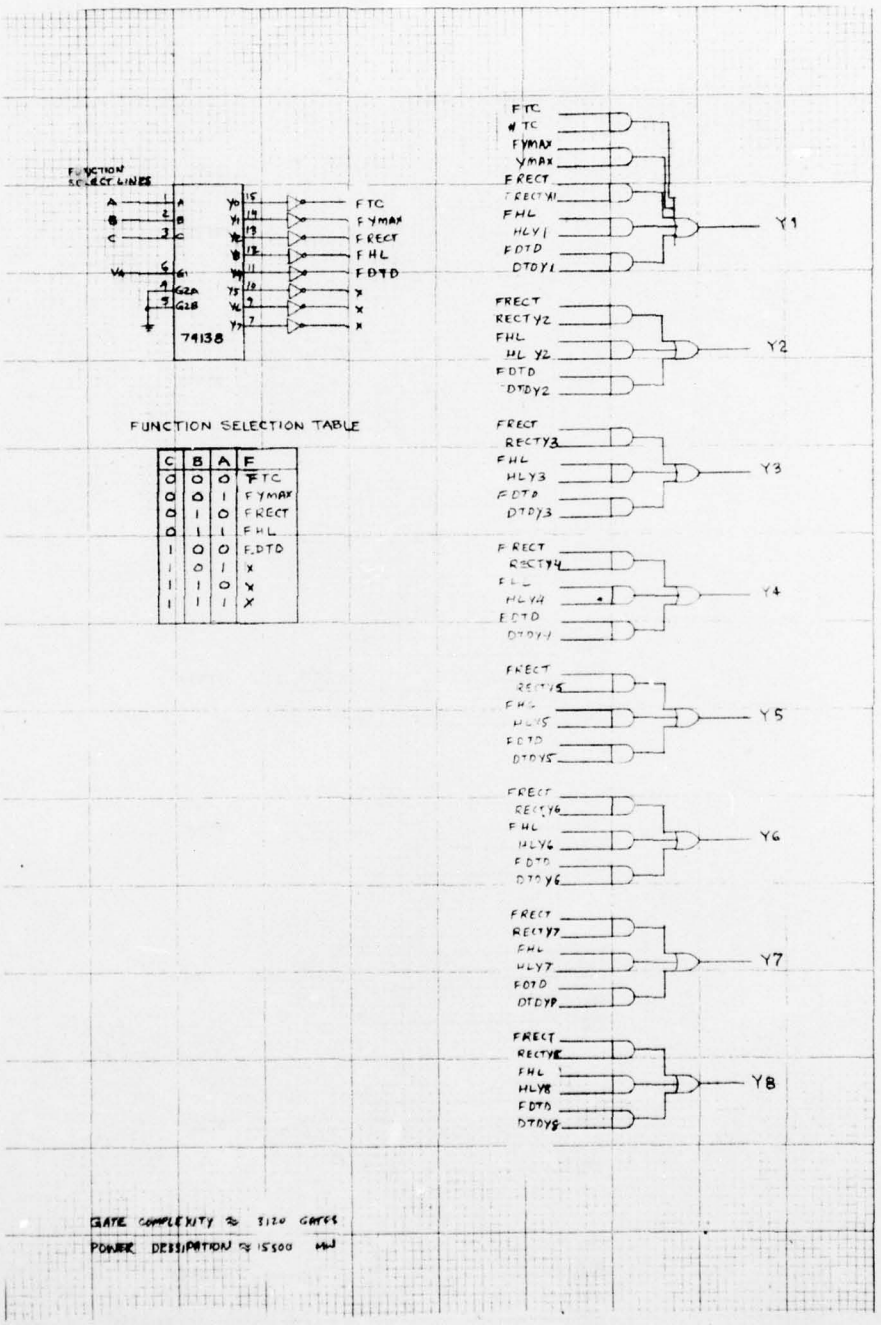


FIG. 70 FUNCTION OUTPUT

SECTION XI

SUMMARY AND CONCLUSIONS

Methods for implementing elements using available LSI technology and capable of efficient realization of a family of multinomial expressions have been explored. These elements, configurable in arrays, represent a new means for the hardware solution to a variety of signal processing problems. Relieving software requirements and providing for a wide range of high speed, real time solutions, the resultant Configurable Polynomial Array (CPA) is applicable to both conventional known linear transformations and processing (such as the FFT) as well as non-linear transformations derived by training algorithms for the realizations of Adaptive Learning Netoworks (ALN's). These CPA's then represent a means for performing all of the critical processing functions associated with classification, discrimination, modeling, identification or recognition type problems. In the performance of such functions, the CPA is much faster than programmed, serial computers while relieving software problems associated with complex signal processing.

Alternative architectural approaches for implementing elements were shown and compared as a function of calculating precisions. These approaches were basically characterized by whether they used all-parallel or serial-parallel multipliers, their control structure (hard-wired or micro-processor), and whether they were to be used in a multiplexed or fully populated array mode.

Supporting precision requirements investigation and analysis showed (by theory and simulations) the application areas of relatively small word lengths (12 to 16 bits, fixed point) up to 32-bit floating point notation.

AD-A042 256 RCA GOVERNMENT COMMUNICATIONS SYSTEMS SOMERVILLE N J --ETC F/G 9/2
LSI ELECTRONICALLY PROGRAMMABLE ARRAYS: CONFIGURABLE POLYNOMIAL--ETC(U)
JUN 77 D HAMPEL, K PROST, R L BARRON F33615-73-C-1089
UNCLASSIFIED AFAL-TR-76-228 NL

3 OF 4
AD A042256

To realize one of the more flexible and efficient element approaches, a 24-bit serial-parallel CMOS/SOS multiplier was designed and fabricated for use in the mantissa processor of an element capable of evaluation of any one of a family of 5 multinomial expressions. Eight of these circuits, along with other T^2L and MOS parts and working in conjunction with an 8-bit floating point processor were assembled into a brassboard for demonstration of the CPA element concept. Related software and hardware, for integrating this brassboard, with an HP21MX was also developed.

The major conclusions of this project are:

1) CPA's are very well suited to signal and image processing applications. Because of their electronic configurability, they can perform nearly all of the bulk arithmetic operations needed in such processing, and when used in distributed-function networks they can provide extremely fast solutions that would require much more time using "general purpose," i. e., predominantly serial, computations.

2) The principal functions for which CPA's should be configurable are primarily 5 expressions (Section II, Table 1).

3) Three nominal CPA architectures have emerged as being the most effective and mutually complementary for realization of CPA functions (Section VI)

    a) A fast, very precise CPA having 5 - 10 us. throughput and using fixed-point arithmetic with 24-bit mantissas, as called for by many real-time signal processing tasks. (A second version of the CPA would incorporate floating-point arithmetic capability). The precision of this CPA is suitable for networks having five to ten layers of parallel distributed functions.

    b) A very fast, moderately precise CPA having 1 - 2 us. throughput and using fixed-point arithmetic with 16-bit mantissas. This precision is suitable for networks having less than five layers of parallel functions

    c) A minimum power consumption, low precision CPA having voltage control of throughput time (to conserve energy) between 10 us. and 1 ms. and using fixed-point arithmetic with eight-bit mantissas. This precision is

suitable for networks having a single layer of parallel functions, if an output accuracy of four bits is sufficient. This CPA also may be used in networks having two (and perhaps up to four) layers if an output accuracy of one bit is acceptable, as in pairwise-voting discriminant functions for signal classifiers.

A signal processing concept in which a configurable polynomial building block can be considered as a "macro" has been demonstrated. The availability of a family of such blocks and their use in arrays will provide improved tools for a variety of Avionics signal processing requirements. The realizations of true parallel array processing embodying distributed logic-memory elements will provide improved problem solving capability in conventional as well as adaptive transformation networks.

Future tasks can now confidently be addressed in the areas of multiple element realizations under a common processor controller to demonstrate previously unattainable throughputs together with ease of software and hardware mechanization for many important signal processing problems.

APPENDIX A

THEORY AND APPLICATION OF CYBERNETIC SYSTEMS: AN OVERVIEW

Roger L. Barron

Adaptronics, Inc.
McLean, Virginia 22101

Abstract

Major aspects of the theory and application of cybernetic systems are summarized in this paper. The principal topics discussed are: methods for synthesis of trainable networks to model relationships in their environment, self-organizing control techniques, guided random search (optimization) procedures that adjust parameters in adaptive systems, hardware trends, and results of representative applications. An attempt is made to present past and current developments in their historical context.

1. Introduction

Cybernetic systems are goal directed, employing information feedback to assess the degree of goal attainment. They are ultrastable processes, that is, these systems become adapted to a problem environment by acting selectively toward responses of important variables observed as internal parameters of the system are changed. [8,35]

Realization of major cybernetic systems involves a number of important subject areas, chief among them being the modeling, control, and adaptation of complex processes. Traditionally, in theoretical science and engineering, these topics have been dealt with by reasoning directly from underlying physical principles, deriving equations that express the analytical conclusions. These conclusions have then been compared with experimental findings, and where agreement is lacking there has been a stimulus for new advances in the theory. It is often taught that this "scientific method" holds the key to solution of most technological problems.

However, it is being found that complex processes are not entirely tractable when approached in the traditional way. It need not be argued that the "scientific method" is wrong, but only that its domain of employment has been too narrow. Technologists have mistakenly assumed that "being scientific" meant achieving closed-form analytical representations for all processes of interest. This may be true in pure science, but in most other endeavors the main object is to create systems that work. And in these endeavors, the old luxury of treating analytically tractable problems can lead to failure.

Neither is it entirely safe to rely on laboratory experimentation. The artificial environment does not always replicate the complexities of the real world.

Faced with the challenge of modeling, control, and adaptation of complex processes, it seems that our paradigm must be more than a method contrived by man's brain, it must be the operation of the brain (or central nervous system) itself. The brain, one of Nature's more remarkable designs, is an existence proof that ultrastability can work in complex processes.

Cybernetics is the science of how the brain performs and how its workings can be realized in the inanimate parts of systems.

The theory of cybernetic systems is still fragmentary, although it is gradually being placed on a rigorous footing, particularly by Ivakhnenko, Rastrigin, and other investigators in the Soviet Union. Even though the functioning of the brain and procedures for copying this functioning in physical systems are only partially perceived, there is presently very rapid progress in applications work. Many of the things being done today were only talked about ten years ago. Cybernetics is beginning to make an indelible imprint on engineering systems. The overview presented here is an attempt to record the most important things that are happening and to identify the trends in these events.

2. Early History of Cybernetics

Refs. 1-18 are milestones in the early development of modern cybernetics theory. An essential thread woven through all of these references is that deterministic rules of behavior, while they sometimes apply in the large, do not appear to govern microscopic levels of activity in problem solving by the brain. At these levels, the laws of chance generally dominate,

with subtle biasing of probabilities being the principal mechanism for guidance of the brain's responses.

The brain is endowed with immense variety of behavior, this richness of activity apparently being the result of probabilistically-controlled internal states. Thus, depending on bias levels within its neurons and neuron aggregates, the brain can realize a highly predictable input-output response at one moment, and at the next instant its behavior can be entirely spontaneous, showing very little obvious relationship to environmental factors.

On the average, the activities of the brain are guided by its objective of goal attainment. Without getting into the psychology of goals and sub-goals, goal selection, and the interplay between conflicting goals, consider how the brain organizes itself vis-a-vis a given goal. Partly by pre-conditioning and partly by selective biasing, the brain establishes data transformations, makes inferences about its environment, decides between alternatives, and initiates actions, largely under the dominance of the goal at hand. This goal can be, in many cases, simply stated, and the degree of goal fulfillment is then simply gaged. Other times the goal is exceedingly complex. But a characteristic of all goals is that their mathematical description is of a much lower dimensionality than the actions needed to fulfill them. In simplest extreme, the rate of goal attainment becomes a scalar measure (e.g., rate of food intake), while the directed activity still has very many variables.

The early engineering applications of cybernetics were rather limited, but the basic principle of probabilistically-controlled states directed via feedback measures of goal attainment, thereby creating rich, one-into-many, dynamic transformations within artificial brains, became well established in the literature of this field by 1960.

3. Recent History of Cybernetics

In retrospect, 1960 was a watershed year in the development of cybernetics. At that time, engineers began to turn to this embryonic science for help in the solution of several challenging problems in the design of physical systems. One of the first problems treated was that of obtaining improved adaptive control of flight vehicles. [Barron in 28, 31; Gwinn and Barron, 34] Another was rapid prediction of re-entry object trajectories. [Snyder, Barron, et al., 25; Gilstrap, 33] Along with work toward these applications, efforts were intensified to place the theory of probabilistically-controlled cybernetic systems on a more rigorous foundation. The work of Lee [21-24] and Gilstrap [21-24,30,33] was particularly notable during the early 1960's.

Contributions by Ivakhnenko [29,38], Gilstrap [36], Mucciardi [35,39], and Barron [37] in the late '60's and early '70's emphasized establishment of a systematic methodology for synthesis of cybernetic systems. This methodology will now be outlined.

4. Modeling via Adaptive Learning Networks

A cybernetic system operates with a goal that is to be fulfilled at a future time. Gilstrap [36] has shown that the types of functions that must be implemented in such a system are a predictor, an objective function for assessment of performance of the system, and a decision rule for selection between alternative actions. The predictor plays a vital role; it anticipates how closely the goal will be satisfied if the system pursues a given plan of action. The predictor thus allows prompt correction of system actions following a disturbance or change in the goal.

Although many forms of predictor are candidates for cybernetic systems, those of greatest utility are the adaptive predictors which learn from past experience. These predictors model future-output vs. present-input relationships for the goal-directed system, using recorded data from past observations. Because the adaptive predictive transformations can be created directly from observations and are not necessarily subject to a priori assumptions, these predictors are potentially as accurate as basic measurements permit.

However, a powerful modeling methodology is required if the full potential of adaptive predictors is to be realized. Multiple linear regression has been tried in many applications, but has been found deficient in the following particulars:

(1) the model must be linear in its unknown coefficients and, therefore, its mathematical structure must be assumed a priori;

(2) the number of data points must exceed the degrees of freedom of the model;

(3) the data structures used cannot be multimodal; and

(4) the least-square-error fitting criterion must be used exclusively

Techniques for nonlinear network modeling have been developed that eliminate these problems. [25,29,30,33,35-38]

The paper by A. N. Mucciardi in this conference is an example of the use of these new techniques. The principal steps in the modeling methodology are:

(1) acquisition of data;

(2) data parameterization;

(3) partitioning of data base;

(4) network training, consisting of selection of most informative parameters, selection of connectivity and approximate coefficient values, optimization of coefficients, and detection and avoidance of data overfitting;

(5) latency reduction of network after computation of its output sensitivities to input variations.

Let us consider each of these steps briefly.

Data acquisition emphasizes recording of all variables that can be economically sensed and that reasonably might be relevant to the modeling task. (Variables not needed will be automatically eliminated later in the synthesis procedure.) Standard rules for the design of experiments are followed in the choices of operating regions, specific conditions within regions, and the number of experiments at each condition. Data sampling rates are chosen to preserve the information content of signals and avoid aliasing.

Data parameterization is performed to compress the dimensionality of the original data and thereby reduce the total synthesis work. Care must be taken not to discard important information. If in doubt, all the original variables should be used. However, when the original measurements describe waveforms of system variables, the derived parameters usually may include conventional quantities such as measures of energies in specific frequency bands, Fourier coefficients, integral functions indicative of waveform shapes, time derivatives, peak-to-peak ratios, mean and rms values, etc. Whereas in linear modeling the extraction of suitable data features is a crucial task (because only the input features of linear models introduce interactions and nonlinear functions of the variables), this problem need not arise in modern nonlinear network modeling: these networks create their own rich combinations of inputs, generating-- in essence -- an additional nonlinear feature set within each of their layers.

Partitioning of parameterized data is done to obtain data subsets for training, testing (overfit avoidance), and independent testing (estimation of expected accuracy) of the network models. A clustering algorithm [39] is used to insure that each of these subsets contains a balanced representation of the total data base: the constituent points within each mode (cluster) of the parameterized data should be distributed in a constant ratio among the subsets. Clustering analysis also reveals if the characteristics of the measured process were stationary over time and if the sensors provided consistent readings.

Network training techniques have never been fully disclosed in the literature, but Refs. 35-38 provide an outline of a procedure that has enjoyed a great deal of success. The basic approach is to synthesize the network with building-block elements, first using a deterministic algorithm [38] to select the best input parameters and to establish the connectivity of the network, then employing a guided random search algorithm to obtain coefficient values that are optimum in the global sense. Mathematically, training is viewed as the realization of a suitabl hypersurface approximation to the training data subset, with the testing subset used to terminate fitting of training data points before overfitting occurs.

The building-block elements most commonly used implement the bivariate function

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2 .$$

Treating all possible pairs of input parameters in succession, the data in the training subset are fitted with this function; i.e., the best $\underline{w}$ is found for each pair of parameters. Then, with reference to the testing subset, the best M pairwise combinations are retained; these provide M new parameters for transformation in a second layer of the evolving network. This second layer is synthesized in a manner analogous to that of the first, and so on for each additional layer. At any layer the growth of the network may be terminated. When this is done, the network output becomes the output of the best element within the last layer, and all elements not required to generate the inputs for this output element are discarded.

In general, the outputs of elements in a given layer, L, are multinomials of degree 2^L involving as many as 2^L input parameters of the network. Despite this geometric growth in network modeling (viz., transformation) power with each added layer, the computational burden increases only linearly with L.

Additional layers are synthesized as long as the performance (fitting accuracy) of the network improves vis-a-vis the testing subset of the data. But when

this performance starts to degrade, it is clear evidence that the training data have been overfitted, i.e., that the network has grown too large. Growth is stopped short of the layer at which overfitting occurs. With this procedure, the network may be trained on a very small amount of data, yet possesses ability to generalize accurately on its limited experience.

Once the deterministic phase of the synthesis is completed, expected accuracy of the network is estimated using the independent testing data subset. Sometimes this accuracy is not sufficient for a given application, indicating that a synthesis phase involving global optimization of the network coefficients (w's) is required. Also, when the network is put into actual service as a predictor, it may be found that its accuracy gradually deteriorates, perhaps because of changes in properties of the process that has been modeled. If this erosion of accuracy is severe, complete re-training might be needed. Usually, however, predictor adaptation involves only the network coefficients, keeping the identities of input parameters and the connectivity fixed. Thus the same type of algorithm may be used in adaptation as in the final phase of synthesis, although newly-acquired data must replace at least some of the data used during initial training. The specific algorithm that has been found to be most suitable in both cases is the Guided Accelerated Random Search (see Section 6 of this paper). When using a search algorithm, just as in the deterministic synthesis phase, improvements noted with respect to a training data subset are tested with a second statistically-similar subset before their acceptance as genuine.

Latency reduction of the network is the last act in the modeling methodology. To this end, the surviving input parameters from the previous steps are again clustered. Centroids, called the prototypes, of the important modes in the data are determined from the cluster structure. These prototypes are representative of the most-frequented operating points of the system. The network inputs are set equal to values given by each of the prototypes in turn, and for each prototype the network output sensitivities to small excursions performed one-at-a-time in its inputs are found numerically. Mathematically, this is the determination of

$$\left(\frac{\partial \hat{y}}{\partial x_i}\right)_{\underline{x}_j} ; \quad \begin{matrix} i = 1, \ldots, N; \\ j = 1, \ldots, K; \end{matrix}$$

where $\hat{y}$ denotes the network output (the estimated future value of a system variable), x_i is an input parameter of the predictive model, and $\underline{x}_j$ is an input prototype vector from the final clustering analysis. If the network has been trained properly, these sensitivities are usually precise reflections of the true physical process. Noting which sensitivities are large in magnitude and which are small, it is often possible to replace many of the x_i's by their respective constants from the $\underline{x}_j$'s, changing these constant inputs only when operating conditions of the system are changed from one mode to another. It is sometimes possible to eliminate sensing of certain of the inputs altogether.

5. Self-Organizing Control Techniques

A characteristic of cybernetic systems is that they usually incorporate a hierarchy of decision and control levels, each having its own appropriate structure of sensors, effectors, and information processors. The goal for a lower level in the hierarchy may often be to implement effectively the commands of the next higher level. Sometimes, however, a difficulty can arise at a given level that pre-empts the attention of the levels above it, forcing them to re-organize their processing logic so as to meet the contingency.

For example, a flight vehicle may nominally have outer guidance loops, within them certain control loops, and internal to all of these a family of stabilization loops (such as vehicle angular position, vehicle angular rate, and actuator displacement and rate loops). Suppose the guidance loop has a predictive model that assumes a certain nominal set of control-loop dynamics will be maintained; if these dynamics undergo change, the predictive model could be re-trained. (The previous section of this paper indicates how such re-training might be accomplished.)

Let us imagine that the control loops of this vehicle normally receive a vector input from the guidance processor. This input consists of three components of the desired translational acceleration. The control processor normally converts this information into orientation commands for the pitch, yaw, and roll stabilization axes, taking into account the actual accelerations to compute its error signals. The control processor may use a model of the transfer properties of the stabilization loops, each of which has the goal of keeping an axis of the vehicle pointed in a commanded direction.

Now, consider the problem of re-organizing the control processor in the event of a contingency arising in the inertial orientation sensing used by the stabilization loops. For purposes of discussion, suppose that a radiation sensor

is used as a back-up in the control subsystem for measuring the magnitude, B, of an off-boresight error angle (but not the direction of this error). Now ask: "If the attitude stabilization loops are lost due to failure of the inertial reference unit, could the control processor change its logic in such a way as to solve the problems of vehicle stabilization and control, given only B(t)?"

At first glance, this illustrative problem may seem to be expecting too much of the control processor, because not only must it deal with a radically different set of dynamics for its controlled object, or plant, after the failure, but it must also generate a 3-vector command (pitch, yaw, and roll actuators) from a single scalar input variable, B(t). Nevertheless, this problem has a solution; the type of control processor that has these capabilities has been the subject of considerable study since the early 1960's.

A reasonably complete picture of the development of self-organizing control techniques, suitable for the above problem and for many other applications (usually for primary rather than back-up control), is provided by Refs. 29, 34, and 40-80. However, what these references do not make entirely clear is that self-organizing controllers (SOC's) are now being widely applied in communications systems, antenna control, electronic warfare, steelmaking, injury recovery control systems, and other areas in addition to flight vehicle control. Also, reading the references, it is easy to lose sight of the essential unity between the fields of adaptive prediction, self-organizing control, and guided random search. Specifically: (1) a multivariate SOC usually employs a form of guided random search that acquires information about the process being controlled, (2) SOC's generally act to control predicted future states of process responses.

This unity is borne out in one ongoing industrial application [77] where SOC logic has been merged with predictive networks to create an extremely powerful cybernetic system. Both the SOC logic and the adaptive networks in this system use guided random searches, but the frequency of adaptations within the SOC logic (which repeatedly interrogates the networks) is considerably higher than in the networks. This illustrates a further principle: in hierarchical cybernetic systems, adaptation processes usually take place most rapidly in the innermost decision and control loops and progressively less rapidly as one moves outward. Ultimately, the level of an inflexible objective function (goal) is reached; this outermost function, which may or may not be explicit, is not adaptive.

Two avenues of SOC development have been followed. In both, the SOC identifies the values, or at least the signs, of members of the open-loop actuation gain matrix of the controlled plant, using this information to realize satisfactory control. Both approaches employ small random excitations so that, through observation of plant responses, the requisite matrix information can be acquired. Rates of change of plant responses are monitored and correlations between excitation signals and these responses are computed to identify polarities and/or magnitudes of the gain matrix elements.

In the original approach, described in Refs. 34,47-49, 53, 55, 57-62, 65-69, 71-73, and 75, the SOC logic is partitioned into two types of modular devices. The first, referred to as a performance assessment unit, computes a binary value signal indicative of the trend in error performance of the system. This value signal is the logical product of the sign of a component of the predicted error and the sign of the appropriate component of the rate of change of plant acceleration. When acceleration is changing in such a way as to reduce the predicted error of the system, the value signal becomes positive; otherwise, it is negative. The other modular function, referred to as the actuation-correlation logic unit, generates a component of the control excitation signal by forming the product of a function of a component of the predicted error signal and a high-frequency polarity decision signal. The decision signal, updated at a frequency considerably higher than the natural frequency of the plant, is itself the output of a statistical experiment generator that is biased by a function of the computed short-term cross-correlation between the received value signal and the polarity signal from the prior decision time. The SOC modules are combined in a controller for which the inputs are the desired and measured responses of the system and from which the outputs are the excitation signals to plant actuators. For the case of a plant having m actuators and n commanded response variable, $m \geq n$, the original SOC employs n performance assessment units and m x n actuation-correlation logic units, each of the latter corresponding to one member of the acceleration gain matrix.

The original SOC logic designs identify polarities of the acceleration gain matrix elements. A recent derivation by D. Cleveland and L. O. Gilstrap, Jr. [76, 78] provides a mathematical foundation for identification of both the magnitudes and polarities of these quantities, and this newest technique has also been realized in working hardware [78].

6. Guided Random Search Procedures

The preceding discussion and the literature it references bear out the thesis that parameter searches are required by adaptive cybernetic systems so as to realize self-adjustment of their behavior. It is often necessary that these searches solve a global optimization problem, i.e., locate the best extremum from among multiple performance modes (hills or valleys). In nearly all cases rapid convergence is required in a noisy signal environment and for simultaneous (or nearly simultaneous) adjustment of multiple parameters.

Several types of search logic are treated in the literature. The major types are systematic, gradient (steepest descent) and random searches. [25,30,33-37,49,60, 90,94,98,100,115]

Systematic searches employ an exhaustive survey of all possible parameter values and are, therefore, capable of finding the global extremum of a multimodal performance function. The drawback to this type of search is that it can be much too time consuming in practical applications. Also, the systematic search is overly noise sensitive and may have an unacceptable search loss (low average performance during the search) because it does not dwell principally in regions of good performance, unless such regions encompass most of the parameter space.

Gradient searches are often an effective way for seeking a local extremum, particularly for spaces of low dimensionality, but are incapable of solving the multimodal search problem unless coupled with other methods. Gradient searches may also have other drawbacks, as implied in the following discussion.

Random searches [82-86,88-115] are of two basic types, unguided and guided. Unguided random searches are a form of exhaustive search, but with a random strategy for trial generation. Guided random searches achieve enhanced rates of convergence via probabilistic control and certain heuristic principles, as will be discussed. Random searches of both types are inherently suited for multimodal search and optimization problems; however, only the guided random searches are of practical interest in most engineering applications.

A particularly powerful form of guided random search, known as the Guided Accelerated Random Search (GARS) [30,33-37, 60,77,91,100,114,115], contains two phases, with control of the search switched back and forth between these phases as certain events occur. In the random phase, an information gathering phase, values of parameters in the search are chosen at random but subject to a multivariate probability distribution function (pdf) that governs the relative amounts of time spent in various regions of the parameter space. In the deterministic phase, information acquired from the random experimentation is exploited in accordance with appropriate heuristic rules.

Whereas unguided random searches employ uniform pdf's to govern a selection of all trial values of the parameters, the uniform distribution is used only in the opening stage of GARS. Afterward the pdf is shaped to hasten convergence to local solutions and, ultimately, the global solution. Then, as the search nears completion, the number of parameters being simultaneously searched is sometimes lowered from the total number of parameters in the search to some fraction thereof, selected at random for each trial.

The random phase of GARS may comprise distinct strategies suitable for the opening, middle, and final stages of the search. In the opening stage, a uniform pdf governs search trials, so as to conduct the exploration with equal probability throughout the parameter space.

In the middle stage, two techniques exist in GARS by which the standard deviation of random steps may be reduced so as to guide, i.e., quicken, the further trials. The first of these techniques produces an explicit identification of the modes of the parameter space. For this purpose, results of the opening search stage are clustered numerically so as to reveal regions of highest and lowest performance and to characterize these regions in terms of their centroids and variances with respect to each of the parameters.[114] The second technique selects some best fraction of the trials conducted in the opening stage and initiates further random trials that are bunched about the points in that fraction. These further trials are conducted with a relatively small step-size standard deviation and have, as their purpose, identification of the modal structure of the performance surface.

The final stage of GARS is entered when the best-to-date performance reaches a level near to the asymptotic performance expected of the search. In this stage, the step sizes may be controlled partly by the measured best-to-date performance in such a way that the magnitudes of search steps shrink to very small values as peak performance is neared. Additionally, an "activity factor" is sometimes used in the final stage as a means for reducing the dimensionality of the random perturbations, ultimately converting the final-stage search from a simultaneous exploration involving all parameters to a more nearly sequential search involving a small fraction of the parameters in any

one step. This fraction is randomly selected for each new trial.

In most applications, noise in measurements of system performance can produce deceptively good results for any one experiment. To avoid the risk of spurious measurements locking the search on a false solution, the random phase of GARS periodically re-examines the performance it achieves at the supposed best-to-date setting of parameters. Each new measurement of performance for this setting is used to refine the estimate of best-to-date performance. Furthermore, to compensate for non-stationarity of the performance surface, control of the search is returned periodically to the logic for the previous stage.

A useful heuristic principal used in the random phase of GARS is called "reversal." Whenever a random experiment does not produce a performance improvement (as indicated relative to the prior best-to-date performance), a step of equal magnitude in the opposite direction is taken (initially using the best-to-date parameter values to define the point of departure). This step in the opposite direction is used in the expectation that a performance improvement will often be found by moving exactly opposite to an unsuccessful direction.

The deterministic phase in GARS, used in alternation with the random phase, is entered whenever a new best-to-date performance value is obtained. In the deterministic phase, additional basic heuristics that speed convergence are employed. These heuristics act to exploit the information gained from a successful random experiment. The heuristic principles used include:

(1) Repetition -- A successful step is followed by another in the same direction.

(2) Acceleration -- The magnitudes of further steps are lengthened in an arithmetic or geometric progression. This produces rapid progress even though the initial successful step may have been quite small. Then, whenever an accelerated step produces an unsuccessful outcome, the search "backs up" to establish the approximate location of the maximum or ridge that has been traversed. The deterministic phase is exited once this deceleration is completed.

Rapid convergence is important for search techniques used in cybernetic systems. Rastrigin [89] has shown that the mean rate of convergence of a random-direction search will exceed that of a fixed-step-size steepest descent procedure when the criterion function is unimodal and the number of parameters exceeds four. An adaptive-step-size random search has been shown by Schumer and Steiglitz [101] to be even more efficient. They show that for gradient methods the number of criterion evaluations increases in proportion to the square of the number of parameters (N) and the computation time increases as N^3, while for the adaptive-step-size random search the number of evaluations increases in proportion to the first power of N and the computation time as N^2. Gilstrap [cited in 60,100] states that the expected number of criterion evaluations for GARS, divided by the number for a fixed-step-size steepest descent method, is approximately

$$\frac{1}{2N} \log_2 N$$

for $N \geq 3$.

In summary, the advantages of guided random searches such as GARS are:

(1) the search is multimodal, i.e., convergence is independent of initial conditions;

(2) very rapid convergence can be achieved, even for a large number of parameters, making real-time adaptation feasible for major systems;

(3) there are no difficulties with step-size control, gain factors, and the like;

(4) the search is relatively insensitive to noise corruption of the performance criterion measurements; and

(5) any computable performance criterion may be used.

7. Applications and Hardware Trends

The engineering applications of cybernetic systems are, in a very literal sense, nearly as old as technology. (For example, Hammurabi, king of Babylon in the 20th century B.C., devised a feedback mechanism that regulated the water level in an irrigation system.) But until about ten years ago, there was little conscious employment of modern cybernetics theory to aid in realization of goal-directed systems and exploitation of these systems in difficult modeling, control, and optimization tasks. Today, however, the catalog of applications receiving active attention (within the modern context) is large and is growing rapidly.

The accompanying table and the cited references list only those projects of which the author has knowledge; no claim can be made for the completeness of this enumeration. It is seen that modern cybernetic systems are serious candidates in numerous areas of application. And hardware for these new systems has been or is being developed in many areas.

SUMMARY OF APPLICATIONS OF CYBERNETIC SYSTEMS

Area/Application	References *			Hardware Status**
	ALN	SOC	GRS	
Prediction and Forecasting				
Re-entry Vehicle Trajectory Prediction	28,33		28,33	
River Basin Bacteria-Count Forecasting	x			
Airport Visibility Forecasting	x			
Parameter Inference				
Re-entry Vehicle Ballistic Coefficients			100	
Aircraft Parameter Identification			107	
Surface Finish Quality in Precision Machining	126,131			B
Marine Biomass in Fresh Water Ponds	128			
Atmospheric Refractive-Index Gradients	122 (x)			
Waveform Analysis and Processing				
Discrimination between Earthquakes and Remote Nuclear Events	133			
Ultrasonic Nondestructive Testing	130			
Language Identification	135			B
Remote Sensor Systems	x			
Emitter Classification	x			P
Adaptive Filtering; Nonlinear Filtering	33			
Adaptive Computation				
LSI Electronically-Programmable Logic Arrays	30,37,118, 124,137			B
Airframe and Propulsion Avionics				
Aircraft Fly-By-Wire Flight Control		34,43,48-50, 54,58,60,61, 71,120		P
Tactical Missile Flight Control		66,69,79,129		
Remotely Piloted Vehicle Control	73	73,76,78		B
Turbopropulsion System Control	117,120	34,66,68,117	117,120	B
Spacecraft Orientation and Auxiliaries Control				
Control of Acquisition; Stabilization		44,47,49, 63,66,68-80		B
Energy Allocation	116	118		B
Antenna System Control				
Az, El Control of Dish Antennas		75,78		B
AMTI Phased-Array Immobilization and Processing		115,134,138, 139	134,138,139, 113,121	B
Sonobuoy Communications		80	80	B
Electronic Warfare				
Adaptive Electronic Countermeasures		x		P
Adaptive Electronic Support Measures	x	x	x	P
Process Modeling and Control				
Steel Hot Strip Mill Finishing Speed	119,132	119	119	P
Steel Hot Strip Mill Runout-Table Cooling Sprays	119,120,132, 140	119,120	119,120	
Steel Basic Oxygen Furnace	77,132	77,126,138	77,132	O
Vehicle Exhaust Emissions	x			B
Biomedical Modeling and Control				
Evaluation of Psychotropic Drugs	123,127,138		123	
Analysis of Head Injury Data	x		x	
Head Injury Patient Recovery Control	x	x	x	

*The symbol "x" denotes no publication in open literature. **B=Breadboard, P=Prototype, O= Operational

Aircraft fly-by-wire SOC logic was successfully flight tested by AFFDL and AFAL in 1969.

AFAL brassboard flight tests of adaptive ECM equipment are taking place at the present time, and AFAL is also currently sponsoring development of brassboard adaptive ESM equipment.

AMRL has successfully demonstrated breadboard SOC logic developed for the RPV man-machine interface.

A. E. Zeger presents a paper at this conference on the AMTI phased array adaptive antenna immobilization system, now entering a breadboard development stage under AFAL sponsorship. This system incorporates a form of GARS search in SOC logic to minimize Doppler clutter spread of the radar signals.

One particularly important hardware trend is toward custom large-scale integration of digital circuits for adaptive learning networks. D. Hampel reports on this AFAL project in a paper at the present conference. Hardware development of adaptive learning networks has also been supported by Armco Steel Corporation. Armco is using predictive control via an adaptive learning network for the finishing of hot strip steel and is developing an application for control of an important melting process.

8. Concluding Remark

An overview of the theory and application of cybernetic systems has been presented. Because of security restrictions and the proprietary nature of many aspects of this new field, and also because of space limitations, it has been necessary to omit many significant details. It is hoped, nevertheless, that this survey has conveyed a feeling for the promise and vitality of work that is taking place in cybernetic systems.

9. Acknowledgments

The author gratefully acknowledges guidance and support received from many sources, particularly Mr. C. W. Gwinn, AFAL; Dr. H. L. Oestreicher, AMRL; Dr. R. E. Boni, Armco Steel Corporation; Mr. W. H. Wilson, Telcom, Inc.; Mr. D. Hampel, RCA Corporation; and Mr. A. E. Zeger, General Atronics Corporation.

The author likewise expresses appreciation to his colleagues at Adaptronics, Inc., among them Mr. D. Cleveland and Dr. A. N. Mucciardi. The contributions of Mr. R. J. Lee and Mr. L. O. Gilstrap, Jr. in earlier phases of the Adaptronics work are also singled out for special mention.

10. Bibliography

10.1 Basic Principles (Pre-1960)

[1] Russell, S. B., "A practical device to simulate the working of nervous discharges," J. Animal Behavior, Vol. 3, 1913, pp. 15-35.

[2] Weiner, N., A. Rosenblueth, and J. Bigelow, "Behaviour, Purpose, and Teleology," 1943. (Appears in Modern Systems Research for Behavioral Scientists, W. Buckley (Ed.), Aldine Publ. Co., 1968, pp. 221-226.)

[3] McCulloch, W. S., and W. Pitts, "A logical calculus of the ideas immanent in nervous activity," Bull. of Math. Biophys., Vol. 5, 1943, pp. 115-133.

[4] Householder, A. S., and H. D. Landahl, Mathematical Biophysics of the Central Nervous System, Principia Press, Inc., Bloomington, IN, 1945.

[5] Weiner, N., Cybernetics, John Wiley & Sons, Inc., New York, 1948.

[6] Hebb, D. O., The Organization of Behavior, John Wiley & Sons, Inc., New York, 1949.

[7] Walter, W. G., "A machine that learns," Sci. American, Vol. 185, No. 2, Aug 1951, pp. 60-63.

[8] Ashby, W. R., Design for a Brain, John Wiley & Sons, Inc., New York, 1952.

[9] Farley, B. G., and W. A. Clark, "Simulation of self-organizing systems by digital computers," IRE Trans. on Inform. Theory, Vol. PGIT-4, 1954, pp. 76-84.

[10] Ashby, W. R., Introduction to Cybernetics, John Wiley & Sons, Inc., New York, 1956.

[11] Uttley, A. M., "Conditional probability machines and conditional reflexes," Automata Studies, C. E. Shannon and J. McCarthy (Eds.), Princeton Univ. Press, Princeton, NJ, 1956, pp. 253-276.

[12] von Neumann, J., "Probabilistic logics," Automata Studies, op. cit., pp. 43-98.

[13] _____, The Computer and the Brain, Yale Univ. Press, New Haven, CN, 1958.

[14] Rosenblatt, F., The Perceptron, Cornell Aeronautical Laboratory Rept. VG-1196-G-1, Jan. 1958.

[15] Newell, A., J. C. Shaw, and H. A. Simon, "Chess-playing programs and the problem of complexity," IBM J. Res. and Dev., Vol. 2, Oct 1958, pp. 320-335.

[16] Barlow, H. B., "Sensory mechanisms, the reduction of redundancy, and intelligence," Proc. Symp. on Mechanization of Thought Processes, Vol. 2, 1958, pp. 535-559.

[17] Lee, R. J., Self-Programming Information and Control Equipment, Melpar, Inc., Falls Church, VA, 1959.

[18] _____, "Generalization of Learning in a Machine, 14th Nat'l. Meeting of Assoc. for Computing Machinery, Sep 1-3, 1959.

10.2 Basic Principles (Post-1960)

[19] Rashevsky, N., Mathematical Biophysics, Dover Publ., New York, 1960.

[20] Campbell, D. T., "Blind variation and selective survival as a general strategy in knowledge processes," Self-Organizing Systems, M. C. Yovits and S. Cameron (Eds.), Pergamon Press, New York, 1960.

[21] Gilstrap, L. O., Jr. and R. J. Lee, "Learning Machines," Proc. Bionics Symposium, USAF WADD Technical Report 60-600, Sep 1960, pp. 437-450.

[22] _____, "Systems that learn," Human Factors in Technology, E. Bennett, J. Degan, and J. Spiegel (Eds.), McGraw-Hill Book Co., Inc., New York, 1963, pp. 459-462.

[23] _____, and M. J. Pedelty, "Learning automata and artificial intelligence," Human Factors in Technology, op. cit., pp. 463-481.

[24] Lee, R. J., and L. O. Gilstrap, Jr., R. F. Snyder, and M. J. Pedelty, Theory of Probability State Variable Systems (Six Vols.), Adaptronics, Inc., Final Technical Report under Contract AF 33(657)-7100 for Air Force Avionics Laboratory, ASD-TDR-63-664, Dec. 1963.

[25] Snyder, R. F., R. L. Barron, et al., Advanced Computer Concepts for Intercept Prediction, Vol. I: Conditioning of Parallel Networks for High-Speed Prediction of Re-entry Trajectories, Adaptronics, Inc., Final Technical Report under Contract DA-36-034-AMC-00099Z, Nike-X Project Office, Redstone Arsenal, AL, Nov 1964.

[26] Gwinn, C. W., "The scope and methods of engineering bionics," Scientia, Vol. 100, No. 246, 1965.

[27] Beer, S., Decision and Control, John Wiley & Sons, Inc., New York, 1966.

[28] Oestreicher, H. L., and D. R. Moore (Eds.), Cybernetic Problems in Bionics (1966 Bionics Symposium), Gordon & Breach Science Publishers Inc., London, 1968.

[29] Ivakhnenko, A. G., Kiberneticheskiye Sistemy s Kombinirovannym Upravleniyem (Cybernetic Systems with Combined Control), Izdatel'stvo Tekhnika, U.S.S.R., 1967.

[30] Gilstrap, L. O., Jr., H. J. Cook, and C. W. Armstrong, Study of Large Neuromime Networks, Adaptronics, Inc. Final Technical Report under Contract AF 33(615)-5125, Air Force Avionics Laboratory, AFAL-TR-67-316, AD # 824 470, Dec 1967.

[31] von Gierke, H. E., W. D. Keidel, and H. L. Oestreicher (Eds.), Principles and Practice of Bionics, AGARD Conf. Proc. No. 44 (1968), Technivision Services, Slough, England, 1970.

[32] Sommerhoff, G., "The abstract characteristics of living systems," Systems Thinking, F. E. Emery (Ed.), Penguin Books, Inc., Baltimore, 1969.

[33] Gilstrap, L. O., Jr., "An adaptive approach to smoothing, filtering, and prediction," Proc. 1969 NAECON, Dayton, Ohio, May 1969, pp. 275-280.

[34] Gwinn, C. W., and R. L. Barron, "Recent advances in self-organizing and learning controllers for aeronautical systems," Proc. AGARD Symp. on Application of Digital Computers to Guidance and Control, London, Jun 2-5, 1970.

[35] Mucciardi, A. N., "Neuromime nets as the basis for the predictive component of robot brains," Cybernetics, Artificial Intelligence, and Ecology, Robinson and Knight (Eds.), Spartan Books, 1972, pp. 159-193. (Presented at 4th Ann. Symp. of Amer. Soc. for Cybernetics, Washington, D. C., Oct 1970.)

[36] Gilstrap, L. O., Jr., "Keys to Developing Machines with High-Level Artificial Intelligence," ASME Paper 71-DE-21. (Presented at ASME Design Eng'g. Conf., New York, Apr 19-22, 1971.)

[37] Barron, R. L., "Adaptive transformation networks for modeling, prediction, and control," Proc. IEEE/ORSA Joint Nat'l. Conf. on Major Systems, Anaheim, CA, Oct 25-26, 1971.

[38] Ivakhnenko, A. G., "Polynomial theory of complex systems," IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-1, No. 4, Oct 1971, pp. 364-378.

[39] Mucciardi, A. N., "An automatic clustering algorithm and its properties in high-dimensional spaces," IEEE Trans. on Systems, Man, and Cybernetics, Vol. SMC-2, No. 2, Apr 1972, pp 247-254.

10.3 Self-Organizing Control

[40] Aseltine, J. A., A. R. Mancini, and C. W. Sarture, "A survey of adaptive control systems," IRE Trans. on Automatic Control, Dec 1958.

[41] Proceedings of the Self Adaptive Flight Control Systems Symposium, P. C. Gregory (Lt., USAF) (Ed.), Wright-Patterson Air Force Base, OH, Jan 13-14, 1959, Wright-Air Development Center TR 59-49, Mar 1959.

[42] Dommasch, D. O., and R. L. Barron, The Theory and Application of Optimum Limited-Information Adaptive Flight Control Techniques, Dodco, Inc., WADC TR 59-468, Oct 1959.

[43] Ostgaard, M. A., and L. M. Butsch, "Adaptive and self-organizing flight control systems," Aerospace Engineering, Sep 1962.

[44] Lee, R. J., and R. F. Snyder, Functional Capability of Neuromime Networks for Use in Attitude Stabilization Systems, Adaptronics, Inc. ASD-TDR-63-549, 1963, AD #429 116.

[45] Morgan, B. S., Jr., "The synthesis of linear multivariable systems by state variable feedback," Proceedings JACC, Stanford, CA, 1964.

[46] Rekasius, Z. V., "Decoupling of multivariable systems by means of state feedback," Proceedings Third Allerton Conference, Montecello, IL, 1965.

[47] Barron, R. L., et al., Self-Organizing Spacecraft Attitude Control, Adaptronics, Inc., AFFDL-TR-65-141, 1965, AD #475 167.

[48] _____, _____, Self-Organizing Control of Aircraft Pitch Rate and Normal Acceleration, Adaptronics, Inc., AFFDL-TR-66-41, 1966, AD #801 157.

[49] _____, "Self-organizing and learning control systems", AD #811 244, in Cybernetic Problems in Bionics [Ref. 28], pp. 147-203.

[50] Smith, F. B., Jr., et al., Logic Networks for Flight Control Applications, Honeywell Inc., AFFDL-TR-65-216, May 1966.

[51] Boskovich, B., and R. E. Kaufmann, "Evolution of the Honeywell first-generation adaptive autopilot and its application to F-94, F-101, X-15, and X-20 vehicles," AIAA J. Aircraft, Vol. 3, No. 1, Jul-Aug 1966, pp. 296-304.

[52] Hoppe, S. G., H. P. Semmelhack, and C. W. Sivonger, A Feasibility Study of Self-Learning Adaptive Flight Control for High Performance Aircraft, Honeywell Inc., AFFDL-TR-67-18, Feb 1967.

[53] Barron, R. L., et al., Analysis and Synthesis of Advanced Self-Organizing Control Systems, Adaptronics, Inc., AFAL-TR-67-93, Apr 1967, AD #813 918.

[54] _____, and R. M. McKechnie, III, "Design principles for self-organizing control system flight hardware", Proc. 19th Ann. NAECON, May 1967, pp. 465-473.

[55] Barron, R. L., et al., Synthesis of a Spacecraft Probability State Variable Adaptive Control System, Adaptronics, Inc., Final Project Report, NASA Goddard Space Flight Center, Jun 1967.

[56] Anon., F-111 Self-Adaptive Flight Control System, General Dynamics Corp. Report FZE-12-161, Jun 9, 1967.

[57] Barron, R. L., Self-Organizing Controller (Mark V), U. S. Patent No. 3,519,998, Jul 7, 1970; Re. 671,743, Sep 19, 1967.

[58] _____, "Self-organizing control: the next generation of controllers", Control Engineering, Part I: "The elementary SOC," Feb 1968, pp. 70-74; Part II: "The general purpose SOC," Mar 1968, pp. 69-74.

[59] _____, Analysis and Synthesis of Advanced Self-Organizing Control Systems - II, Adaptronics, Inc., AFAL-TR-68-236, Sep 1968, AD #840 295.

[60] _____, "Adaptive flight control systems", in Principles and Practices of Bionics [Ref. 31], pp. 119-167.

[61] Lee, Y. S. and T. W. Toivanen, Application of [Adaptronics] Mark III SOC to Multivariable Control Problems: Part IV--Optimal Decoupling Control Applied to the Lateral-Directional Axes of a Sweep-Wing Aircraft, Honeywell Inc., AFFDL-TR-68-10, Sep 1968.

[62] Barron, R. L., Self-Organizing Controller with Constrained Performance Assessment, U. S. Patent No. 3,591,778, Nov 5, 1968.

[63] Gilbert, E. G., "The decoupling of multivariable systems by state variable feedback," SIAM J. Control, Vol. 7, No. 1, 1969.

[64] Falb, P. L. and W. A. Wolovich, "On the decoupling of multivariable systems," SIAM J. Control, Vol. 7, No. 3, 1969.

[65] Barron, R. L.,et al., Application of Self-Organizing Control Techniques to High-Performance Missiles, Adaptronics, Inc., Final Technical Report, NASC, Apr 1969, AD #852 869.

[66] _____, _____, Self-Organizing Control of Advanced Turbine Engines, Adaptronics, Inc./ Hamilton Standard Division of United Aircraft Corporation, AFAPL-TR-69-73, Aug 1969, AD #857 616.

[67] _____, Self-Organizing Controller [Mark III], U. S. Patent No. 3,460,096; Aug 5, 1969.

[68] Billig, L. O., "Adaptive controls," Instrument and Control Systems, Part I: "A survey," Vol. 42, Sep 1969, pp. 147-152; Part II: "Self-organizing systems," Vol. 42, Oct 1969, pp. 126-131.

[69] Cleveland, D., R. L. Barron, et al., Research and Development on Self-Organizing Control Systems for Air Launched Missiles, Adaptronics, Inc. Final Technical Report, NASC, Apr 1970, AD #867 918.

[70] Ehlers, H. L., and R. K. Smyth, "Adaptive control applications to aerospace vehicles," AIAA J. Aircraft, Vol. 7, Mar-Apr 1970, pp. 97-116.

[71] Anon., Design, Fabrication, and Flight Testing of Self-Organizing Flight Control System, Adaptronics, Inc., AFFDL-TR-70-77, Jun 1970.

[72] Barron, R. L., and J. R. Gouge, Jr., Redundant Self-Checking, Self-Organizing Control System, U. S. Patent No. 3,593,307; Jul 13, 1971.

[73] Anon., Application of Self-Organizing Control to Remotely Piloted Vehicles, Adaptronics, Inc., ASD XR-72-19, Apr 1972.

[74] Pope, R. E., (Lt., USAF), The Design of Stability Augmentation Systems for Decoupling Aircraft Responses, AFFDL-TR-72-63, Jun 1972.

[75] Barron, R. L., K. S. Kelleher, and G. C. Vieth, Jr., Self-Programming Antenna Tracking System, U. S. Patent No. 3,680,126; Jul 25, 1972.

[76] _____, and R. A. Gagnon (Major, USAF), "Application of self-organizing control to remote piloting of vehicles," Remotely Manned Systems (E. Heer, Ed.), Proceedings of NASA/Cal Tech First National Conference on Remotely Manned Systems, Sep 13-15, 1972, Pasadena, Calif., pp. 409-422. (Publ. 1973)

[77] Mucciardi, A. N., and E. C. Orr, Ashland BOF Learning Control System--Task II: Preparation of Learning Control System Supervisory Logic, Adaptronics, Inc., Final Technical Report, Armco Steel Corporation, Jan 9, 1973.

[78] Cleveland, D., R. L. Barron, J. R. Binkley, Jr., and L. O. Gilstrap, Jr., RPV/Self-Organizing Control Demonstration System: Volume I -- SOC Equation Development, Logic Configurations, and Control Modes; Volume II -- Hardware Description, System Operation and Maintenance, and RPV Simulation, Adaptronics, Inc., AMRL TR-73-66, Jun 1973.

[79] Barron, R. L., and D. Cleveland, Self-Organizing Control System, U. S. Patent No. 3,794,271; Feb 26, 1974.

[80] Baird, C. A., "Search Algorithms for Sonobuoy Communications," paper delivered at NRL Adaptive Antenna Systems Workshop, Washington, D. C., Mar 11-13, 1974.

10.4 Guided Random Search

[81] Box, G. E. P., and K. B. Wilson,"On the experimental attainment of optimum conditions", J. Royal Stat. Soc., Series B., Vol. 13, 1951, pp. 1-45.

[82] Brooks, S. H., "A discussion of random methods for seeking maxima", Operations Research, Vol. 6, 1958, pp. 244-251.

[83] Hooke, R., and T. A. Jeeves, "Comments on Brooks' discussion of random methods," Operations Research, Vol. 6, No. 6, Nov. 1958, pp. 881-882.

[84] Brooks, S. H., "A comparison of maximum seeking methods," Operations Research, Vol. 7, 1959, pp. 430-457.

[85] Rastrigin, L. A., "Extremal control by the method of random scanning," Avtomatika i Telemekhanika (Automation and Remote Control), 1960, pp. 891-896.

[86] Hooke, R., and T. A. Jeeves,"'Direct Search' solution of numerical and statistical problems", J. Assoc. Comp. Mach., Vol. 8, No. 2, Apr 1961, pp. 212-229.

[87] Fletcher, R., and Powell, M. J. D., "A rapidly convergent descent method for minimization," The Computer Journal, Vol. 6, 1963, p. 163.

[88] Karnopp, D. C., "Random search techniques for optimization problems," Automatics, Vol. 1, Pergamon Press, 1963, pp. 111-121.

[89] Rastrigin, L. A., "The convergence of the random search method in the extremal control of a many-parameter system," Avtomatika i Telemekhanika (Automation and Remote Control), Vol. 24, 1963, pp. 1467-1473.

[90] Wilde, D. J., Optimum Seeking Methods, Prentice-Hall, Inc., Englewood Cliffs, N. J., 1964.

[91] Matyas, J., "Random optimization," Avtomatika i Telemekhanika (Automation and Remote Control), Vol. 26, 1965, pp. 246-253.

[92] Moddes, R. E. J., L. O. Gilstrap, Jr., R. J. Brown, R. F. Snyder, J. M. Davies, H. J. Cook,and S. Levine, Study of Neurotron Networks in Learning Automata, Adaptronics, Inc.,Final Technical Report under Contract AF 33(615)-1526 for Air Force Avionics Laboratory, AFAL-TR-65-9, Feb 6, 1965, AD #455 688.

[93] McMurtry, G. J., and Fu, K.-S., "A variable structure automaton used as a multimodal searching technique," Proc. Nat'l. Electronics Conf., 1965.

[94] Leon, A., "A classified [annotated] bibliography on optimization," Recent Advances in Optimization Techniques, A. Lavi and T. P. Vogl (Eds.), John Wiley and Sons, Inc., New York, 1966, pp. 599-649.

[95] Yudin, D. B., "Quantitative analysis of complex systems. II. [random search]", trans. from Tekhnicheskaya Kibernetika, No. 1, 1966, pp. 1-13.

[96] McLaren, R. W., "A stochastic automaton model for the synthesis of learning systems," IEEE Trans. on Systems Sci. and Cybernetics, No. 2, 1966.

[97] Gurin, L. S., "Random search in the presence of noise," trans. from Tekhnicheskaya Kibernetika, No. 3, 1966, pp. 252-260.

[98] Barron, R. L., "Parameter space search techniques for learning automata", Proc. 1966 Bionics Symp., Dayton, OH, May 1966.

[99] Bekey, G. A., A. Sabroff, M. H. Gran, and A. Wong, "Parameter optimization by random search using hybrid computer techniques," Proc. AFIPS Computer Conf., Nov 1966, pp. 191-200.

[100] Barron, R. L., "Inference of vehicle and atmosphere parameters from free-flight motions," AIAA J. of Spacecraft and Rockets, Vol. 6, No. 6, Jun 1969, pp. 641-648. (Paper presented at AIAA Guidance, Control and Flight Dynamics Conf., Huntsville, AL, Aug 1967.)

[101] Schumer, M. A., and K. Steiglitz, "Adaptive step size random search," IEEE Trans. on Automatic Control, Vol. AC-13, Jun 1968, pp. 270-276.

[102] Hill, J. D., "A search technique for multimodal surfaces," IEEE Trans. on Systems Sci. and Cybernetics, Vol. SSC-5, No. 1, Jan 1969, pp. 2-8.

[103] Schmitt, E., Adaptive Computer Algorithms for Optimization and Root-Finding, NTZ-Rept. 6, VDE-Verlag, GmbH, Berlin, 1969.

[104] Pensa, A. F., and G. J. McMurtry, "Gradient biased random search," Proc. IEEE Systems Sci. and Cybernetics Conf., Pittsburgh, PA, Oct 14-16, 1970, pp. 171-173.

[105] Gran, R., "On the convergence of random search algorithms in continuous time with applications to adaptive control," Proc. 9th IEEE Symp. on Adaptive Processes in Decision and Control, Austin, TX, Dec. 7-9, 1970.

[106] White, L. J., and R. G. Day, "An evaluation of adaptive step size random search," IEEE Trans. on Automatic Control, Oct 1971, pp. 475-478.

[107] Parrish, R. V., Parameter Identification Using a Creeping-Random-Search Algorithm, NASA TN D-6533, Dec. 1971.

[108] Zak, Yu. O., and G. P. Kondrashin, "Algorithms for random search among a finite number of prescribed directions," Soviet Automatic Control, Vol. 5, No. 1, 1972.

[109] Berlin, V. G.,"Parallel randomized search strategies," Avtomatika i Telemekhanika (Automation and Remote Control), No. 3, Mar 1972, pp. 60-65.

[110] Lawrence, J. P., III, and K. Steiglitz, "Randomized pattern search," IEEE Trans. on Computers, Apr 1972, pp. 382-385.

[111] Beltrami, E. J., and J. P. Indusi, "An adaptive random search algorithm for constrained minimization," IEEE Trans. on Computers, Sep 1972, pp. 1004-1008.

[112] Opačić, J., "An heuristic strategy for multimodal search," Int. J. Systems Sci., Vol. 4, No. 3, 1973, pp. 485-500.

[113] Rastrigin, L. A., and K. K. Ripa, Avtomatnaya Teoria Sloochanogo Poiska, Izdatel'stvo "Zinatneh", Riga, 1973.

[114] Mucciardi, A. N., "A new class of search algorithms for adaptive computation," Proc. 1973 IEEE Conf. on Decision and Control, San Diego, CA, Dec 1973, pp. 94-100.(To be publ. in IEEE Trans. on Systems, Man,and Cybernetics).

[115] Barron, R. L., "Guided Accelerated Random Search as Applied to Adaptive Array AMTI Radar," paper delivered at NRL Adaptive Antenna Systems Workshop, Washington, D. C., Mar 11-13, 1974.

10.5 Further Literature on Applications and Hardware Trends

[116] Idelsohn, J. M., Centner, and A. Speake, "Application of Bionics to spacecraft energy allocation," Bionics Symp. 1966, Short Paper Preprints.

[117] Mucciardi, A. N., R. L. Barron, et al., Adaptive Optimization System for Maximizing Performance of Variable-Geometry Turbojet Compressor with Unknown Surge Boundaries, Adaptronics, Inc., Technical Report #2 for Pratt & Whitney Aircraft Division, United Aircraft Corp., Apr 1970.

[118] Cleveland, D. "Hardware Realization of Trainable Multivariable Nonlinear Transformations," paper presented at Computer Designers' Conference and Exhibition, Anaheim, CA, Jan 1971.

[119] _____, and E. C. Orr, Program for Adaptive Control of Finishing Stand and Coiling Temperatures on the Ashland Hot Strip Mill, Adaptronics, Inc., Final Technical Report, Armco Steel Corporation, Feb 1971.

[120] Barron, R. L., and C. W. Gwinn, "Applications of Self-Organizing Control to Aeronautical and Industrial Systems," ASME Paper 71-DE-22. (Presented at ASME Design Eng'g. Conf., New York, Apr 19-22, 1971.)

[121] Howard, J. E., G. O. Young, and R. A. Birgenheier, Adaptive AMTI Radar Techniques Investigation, Hughes Aircraft Co.,Final Technical Report under Contract F33615-70-C-1754 for Air Force Avionics Laboratory, AFAL-TR-71-201, Jul 1971.

[122] Barron, R. L., and F. W. van Straten, Procedure for Inferences of Radiowave Refractivity Profiles of the Atmosphere, Adaptronics, Inc. Technical Note ATN-65-1, Jul 14, 1971. (Proprietary)

[123] Mucciardi, A. N. et al., Demonstration of Adaptive Computation Methods for the Analysis of EEG Clinical Data (Measurement of Drug Similarity and Estimation of Effective Dosage Level of the Experimental Compound U-31,889), Adaptronics, Inc. Final Technical Report to Missouri Institute of Psychiatry, 1971.

[124] Anon., Basic and Applied Research in Industrial Process Adaptive Controls: Third Quarterly Progress Report for Armco Steel Corporation (Second Year), Adaptronics, Inc., Jan 1, 1972.

[125] Barron, R. L., Preliminary Analysis of SFQ Machining Data, Adaptronics, Inc. Interim Technical Report #1 for Union Carbide Corporation, Feb 16, 1972.

[126] Mucciardi, A. N., "Information filtering using the CLUSTR algorithm", Proc. Computer Image Processing and Recognition Conf., Univ. of Missouri, Columbia, 1972, Vol. 2, pp. 15-3-1 to 15-3-8.

[127] Itil, T. M., B. Saletu, J. Marasa, and A. N. Mucciardi, "Digital computer analyzed awake and sleep EEG (sleep prints) in predicting the effects of a triazolobenzodiazepine (U-31,889)," Pharmakopsychiat., 5, 1972, pp. 225-240.

[128] Anon., Experimental Study of Trainable Network Calibration for in Situ Biomass Estimates, Addendum to Engineering Specifications for an Instrumentation and Data Reduction System for Fresh Water Ponds Study, Adaptronics, Inc. Final Technical Report for Michigan State Univ., Aug 31, 1971 (Date of Addendum: Jul 31, 1972).

[129] Gilbert, W. H., and D. Cleveland, Tactical Missile Self-Adaptive Controller, Martin Marietta Corporation/Adaptronics, Inc. AFFDL TR-72-27, Aug 72.

[130] Anon., Recommendations Resulting from Study of Adaptive Signal Processing as Applied to Nondestructive Testing, Adaptronics, Inc. Technical Report 675F-1 for Air Force Materials Laboratory, Oct 1972.

[131] Barron, R. L., and R. F. Snyder, Preliminary Analysis of Further SFQ Machining Data, Adaptronics, Inc., Interim Technical Report #2-1 for Union Carbide Corporation, Oct 30, 1972.(Rev. Nov 3, 1972)

[132] Mucciardi, A. N., "Elements of learning control systems with applications to industrial processes", invited paper, Proc. 11th IEEE Conf. on Decision and Control, New Orleans, LA, Dec 13-15, 1972, pp. 320-325.

[133] _____, Demonstration of Capability of Adaptive Learning Networks to Discriminate Automatically between Earthquakes and Nuclear Explosions Based on Seismic Parameters, Adaptronics, Inc. Report ATN-76 to Arms Control and Disarmament Agency, Apr 23, 1973.

[134] Burgess, L. R., and A. E. Zeger, Adaptive Techniques for Radar Control, General Atronics Corp., Final Technical Rept. under Contract F33615-72-C-1842 for Air Force Avionics Laboratory, AFAL-TR-73-394, Nov 1973.

[135] Mucciardi, A. N., et al., Application of Adaptive Learning Networks to Discriminate between Five Spoken Languages, Adaptronics, Inc., Technical Report 683F-1 for RCA Corporation, Dec 1973.

[136] Zeger, A. E., "Adaptive Array AMTI Radar," paper delivered at NRL Adaptive Antenna Systems Workshop, Washington, D. C., Mar 1974.

[137] Hampel, D., "Electronically programmable LSI arrays," 1974 NAECON.

[138] Mucciardi, A. N., "Application of adaptive modeling techniques to evaluation of psychotropic drugs," 1974 NAECON.

[139] Zeger, A. E., "Adaptive array AMTI radar," 1974 NAECON.

[140] Barron, R. L., and J. H. Schunk, "Adaptive Nonlinear Modeling for Predictive Control of Hot Strip Steel Mill Runout Table Cooling Sprays," Control Engineering/Purdue Conf. on Advanced Control Techniques, Purdue Univ., Lafayette, IN, Apr 29-May 1, 1974.

APPENDIX B

Networks that learn are useful in computer-aided design and manufacturing to improve process performance, reduce costs, and decrease the need for engineering analyses. Here's how they learn and how they are applied

Learning Networks Improve Computer-Aided Prediction and Control*

Roger L. Barron

Adaptronics, Incorporated
McLean, Virginia

A quiet revolution is spreading in the way that engineers think about manufacturing processes. In the past, these processes were designed and operated on the basis of either doctrinal or engineering concepts. Analysts and engineers have attempted to write instructions for control computers in the same way that they instruct human operators in how to perform appropriate manual control functions. Thus applications of computer-aided manufacturing during the first two decades of the computer era have consciously or unconsciously taken the form of one-for-one replacements of human operator functions by equivalent computer functions. This concept of replacement has maintained or even reinforced traditional reliance on analytically derived models of processes.

Now, however, speed, memory, and dependability of computers are leading to fresh approaches to modeling and control of manufacturing processes, and a number of exciting new methods are receiving attention, one of which is the "learning network" approach. Networks for computer-aided prediction and control, which may be implemented in computer software or peripheral hardware, can learn to predict trends in a process from the natural data it produces. The process is characterized entirely from its observable variables rather than by a set of theoretical equations; its true characteristics are embodied in the natural stream of data that flows from it. The problem is thus one of generating a process model *from data* rather than one of estimating what the data will be.

At the very least, this way of thinking can augment traditional approaches. The learning network identifies which variables play significant roles in the behavior of a process and shows how these variables interact with each other (usually nonlinearly) in determining just what the process will do. Thus the network points the way toward improved theory for traditional modeling work. Alternatively, the network can model the most uncertain aspects of a process, leaving other, already fairly well understood aspects to theoretical derivations.

Learning networks themselves can infer and predict process behavior very accurately. Unlike most predictive models, errors made by these networks do not necessarily increase cumulatively with increasingly longer forecast intervals. Also, they can adapt to changing process characteristics, keeping themselves up to date without requiring tuning or redesign by human specialists.

Perhaps most exciting, learning networks are able to predict from data that are produced "naturally" by processes—including records of sounds or vibrations, subjective evaluations of product quality, or whatever variables are readily accessible for economical measurement. It is only necessary that the natural variables, taken in concert, contain information for the inferences or predictions to be made.

In other words, computer-aided manufacturing (CAM) systems need not rely on instrumentation of state variables and other conveniences of older theories about control. Often there is no way to instrument such

*This article is based on a paper presented by Mr Barron at the 1975 CAD/CAM Conference of the Society of Manufacturing Engineers, held in Chicago, Ill.

variables, either because the transducers required do not exist or because the variables are physically inaccessible. For example, relationships pertaining to a batch reaction are hardly useful if measuring temperature and composition of material in the reactor is impractical. However, a learning network might control such a reaction by monitoring such variables as acoustic emissions or fluctuations in the temperature of off-gases. In doing so, the learning network more closely resembles human-like abilities for making associations between complex patterns of events—with the speed, accuracy, and attentiveness of a computer.

Learning Networks

Inference and prediction problems involve operations with sensor d a obtained as a result of observing a physical process. The classical approach to designing the computer model has been to use all relevant deterministic or statistical characteristics of the process being observed, along with certain assumptions in design calculations. Very often the designer presumes the structure of the model and merely calculates values of certain parameters. Even if the nature of the observed process changes, the structure of the model often does not change; but the designer adjusts the parameter values in response to measured changes in inputs or outputs.

In many important applications, observable inputs are difficult to describe analytically. The best or even a good structure for the model cannot be determined in advance. In this case, a desirable model structure can adjust to representative inputs. That is, the model is *trainable* in both its structure and parameter values.

Trainability in structure implies the existence of interconnections of similar elementary building blocks in a network. This network is described by a general (usually nonlinear) function of certain input variables called observables. Since little may be known about the characteristics of the observables, network parameters are not known in advance, but are learned as the network is trained with representative inputs. This concept raises new questions about what the structure of the elements of the network should be, how the element parameters should be adjusted, how many elements are necessary, and how they should be interconnected.

Polynomial and Multinomial Approximations

Suppose that the input consists of N observables, x_1, x_2, . . . , x_N, and the output, y, is a scalar quantity whose value is the estimate of a particular property of the input process. In general, y will be a nonlinear function of the x's. Under fairly general conditions, this function of N variables can be expressed in an N-dimensional Maclaurin series:

$$y = a_0 + \sum_{i=1} a_i x_i + \sum_{i=1} \sum_{j=1} a_{ij}\, x_i x_j + \sum_{i=1} \sum_{j=1} \sum_{k=1} a_{ijk}\, x_i x_j x_k + \ldots$$

Although in the most general case, the coefficients are functions of time, underlying characteristics of the x's often do not depend on time, so that the coefficients are constants.

To apply this Maclaurin series, identities of the observables or measurements, x_i, must be known, along with the number of terms in the series needed to provide an acceptable approximation to the desired function—even though this function is itself not known. To determine the observables, all those that are *thought* to have a bearing on the desired output are used at first, and the ones whose trial shows them to be of little use are later discarded. The number of terms is determined adaptively with a trainable nonlinear

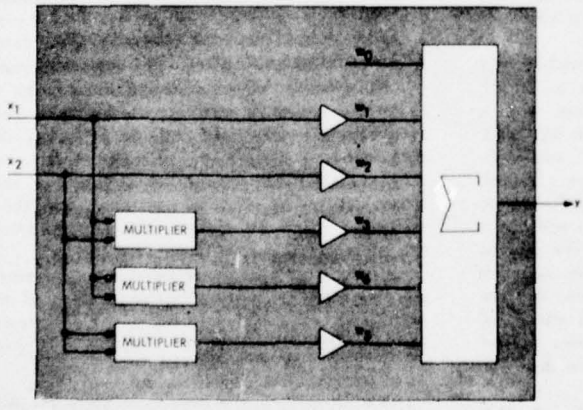


Fig. 1 Learning network element. A digital mechanization of the functions shown here is used to implement the first six terms of a Maclaurin series that expresses a property or characteristic of a process in terms of its inputs. It has only two inputs, but networks of elements such as this can combine all the inputs in pairs. Usually fewer than all possible pairs are necessary

network of interconnected elements, each of which implements a simple second-degree function of two inputs and one output, also including first-degree and constant terms (Fig. 1):

$$y = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2$$

This function is equivalent to the first six terms of the generalized Maclaurin series for two variables. Components of the element may be realized using digital devices. A network of these elements can be trained, by adjusting values of coefficients, to approximate the N-dimensional series.

Networks of the Basic Element

In a network of two layers of these simple elements (Fig. 2), each input, z_i, to the summation contains pairwise products of the network inputs, x_i, up to degree 4, while the first layer contains all possible pairs of three inputs. To implement a general multinomial expression (merely a polynomial in many variables), the number of elements in each layer would have to grow as one proceeds deeper into the network. However, it is found empirically that acceptable approximations are obtained without this growth; in fact, the number of elements in successive layers decreases —usually after only two or three layers—until only a few are left as inputs to the adder.

Known Data Set

Determining the coefficients of each network element and the number and interconnections of those elements requires a known data base—that is, a data base for which the values of the dependent variable are known. Steps involved are (1) optimizing the coefficients in each element of the first layer, (2) selecting those elements whose output is acceptable while rejecting the poor performers, (3) repeating steps (1) and (2) for the remaining layers, and (4) globally optimizing all coefficients in all layers based upon network output.

The known data base is divided into three *independent* but statistically similar subsets: a fitting subset, to determine coefficients of the elements; a selection subset, to reject the poor performers; and an evaluation subset, to evaluate overall performance. Fitting and selection subsets are also used for global optimization. Since the evaluation subset is *not* used for network synthesis, performance on it accurately estimates the network's ability to generalize to new, previously unseen data.

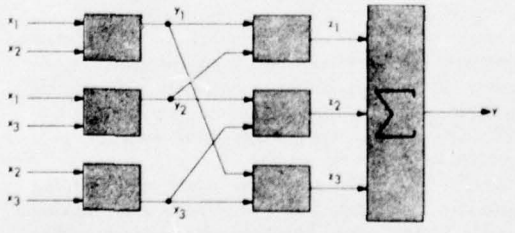


Fig. 2. Two-layer network. Three inputs combine into three pairs at the first level of elements such as that of Fig. 1. At the second level, outputs of the first level recombine in pairs to generate up to fourth-degree signals. Theoretically, later stages would snowball in number and complexity of interconnections, but in practice they do not

Training the Network

Element coefficient determinations are based, in part, upon a least-squares fit to a desired output, whereby the elements are first adjusted by a matrix algebraic procedure and then by a recursive search or optimization procedure. (Other criteria are of course possible, and are often used.)

Fitting and selection subsets are used alternately in training each layer. First, N specific observables that are the inputs to each element are chosen, more or less arbitrarily, and arranged into $N(N-1)/2$ pairs, feeding a like number of trainable elements, such as *that shown in Fig. 1. Then* the fitting subset of the known data base is applied to establish the coefficients, using a recursive search procedure with a least-squares criterion. The procedure is repeated for each of the $N(N-1)/2$ elements.

Not all pairwise combinations are significant in extracting the desired information. The selection process, using the selection subset, eliminates those elements whose performance is not acceptable, as measured by the square of the error magnitude. There are now, say, R elements that survive.

The process is repeated for the second layer, which initially contains $R(R-1)/2$ elements, involving all pairs of the surviving elements in the first layer—which now is again fed by the fitting subset. Coefficients of each element in the second layer are determined as in the first. Then the selection subset is fed a second time into the first layer and the unacceptable pairs eliminated from the second layer.

The process is repeated with succeeding layers until the error rate on the selection subset reaches a suitable low level. Although further reductions in error rate on the fitting subset could be made by incorporating additional layers, to do so would produce overfitting of the fitting data. Eventually, a single output results from each of several disjoint subnetworks; these outputs are added to produce a single output from the entire network.

A hypothetical example of the result of the training process to this point (Fig. 3) implies that at least 30 candidate parameters were initially inserted into the first layer, of which only a few survived. The figure shows that pair (x_1, x_{28}) interacts with pair (x_4, x_{30}), but pairs (x_5, x_8) and (x_{18}, x_{29}) do not interact with each other or with the other pairs. Thus the out-

puts of three disjoint subnetworks are added to produce a single output.

A final step in the training process is a vernier adjustment, or fine tuning, of the coefficients. This may arise because the coefficients of each element have been adjusted in the absence of interactions with other elements following them in the network; optimum coefficient values may be different when these interactions are present. Fitting and selection subsets are also used for this final adjustment process. The vernier adjustment—a global search—may use a random technique to obtain final values of the coefficients, as well as for subsequent network adaptation. After final adjustment of coefficients, the evaluation subset is used to estimate performance of the entire network.

Avoidance of overfitting is a key aspect in the training of learning networks. Good functional approximations to the fitting data subsets must be obtained that also closely approximate the data in the separate selection subsets—that is, the networks can be taught to generalize properly on their experience in fitting the points in the first subsets, so that error rates in later uses will be low. If overfitting is not avoided, the network produces deceptively small errors in approximating its first sets of data and then, in most cases, does poorly on subsequent new data. Often heard of are empirical models that appear to have much promise initially but that produce unacceptable errors when presented with new data: in most cases, such behavior is the result of overfitting. By using three independent subsets of the available data—taking care that each is statistically representative of the whole data base, the problem of overfitting is virtually eliminated and good advance estimates of operational error levels of the models can be obtained.

If a model realized by a learning network can be guaranteed not to be overfitted, it will be a smoothly fitted, functional approximation. Mathematically, this approximation is a continuous and differentiable function, derivatives of which closely approximate the quantitative derivative behavior of the real processes that are modeled. For this reason, numerical partial derivatives may be computed which reveal the quantitative sensitivity of the modeled variable (y)—and thus of the process that has been modeled—to small variations in specified values of the network input variables.

As will be seen, ability to interrogate learning networks at arbitrary points (within their regions of fit to prior data), thus finding predicted values and sensitivities of process responses, is the key to use of these networks in computer-aided design and manufacturing (CAD/CAM).

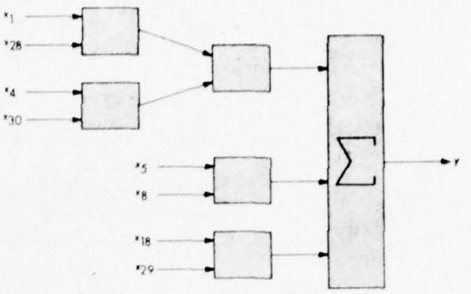


Fig. 3 Illustrative learning network. Although 30 or more inputs were hypothesized as contributing to the process controlled by this network, only eight survived the training process as contributing significantly to the output. Furthermore, only four of the eight required a subnetwork more than one level deep. Typically, learning networks have many more elements than shown here

Use of Learning Networks

A learning network in CAM implements a predictive model for the process being controlled (Fig. 4). A separate network is used for each predicted variable. Inputs to each network are measured process variables and trial control variables. When the switch is in search position, sequence search logic can rapidly interrogate the networks to discover predicted consequences of hypothetical control actions. When the switch is moved to control, the best sequence of control action found by interrogating the networks is transmitted through actuators to the process being controlled.

The sequence may be recomputed as often as desired—most often for a process that is subject to frequent disturbances. Conversely, the more accurate the predictive model, the less frequently the system must recalculate its control decisions.

Sequence search logic is a numerical optimization algorithm whose input is a predicted score computed by performance assessment logic. This score may consist simply of the magnitude of the arithmetic difference between the desired final value of a process variable and its predicted final value, in which instance the goal of the search logic is to find a sequence that drives the score to zero. When multiple variables are to be controlled to specified final values, the score function may be a weighted sum of predicted absolute final errors, in which the more important final variables are given greater numerical weights than the less important variables. Other, more sophisticated score functions may be computed to express such characteristics as quality of steady-state performance, transient recovery from disturbances, or adherence to manufacturing constraints. If flexible search logic such as a guided random search algorithm is used, virtually any computable score function may be employed to govern the search.

While the switch is in search mode, the learning networks are interrogated at a very high rate, making it possible to try many hypothetical sequences within a short period of time. Typically, for software networks having approximately 50 elements, at least 100 interrogations/s can be realized. Achieving a high interrogation rate requires that the networks be efficiently programmed. For the most demanding applications, peripheral computer hardware may also be employed to implement the networks. Depending on the amount of parallel circuitry used in the peripheral hardware, a tenfold-to-thousandfold increase in speed may be realized.

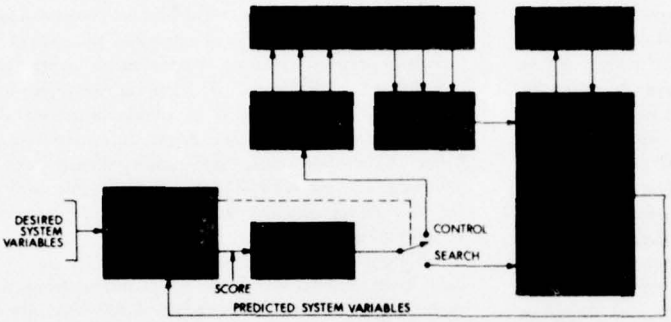


Fig. 4 An application in computer-aided manufacturing. Key elements, shown in color, set up an optimum sequence of control actions in search mode, then apply these actions to the process when the switch is thrown to control mode. If necessary, the system can return to training mode for minor retraining from time to time

When the predicted score is satisfactory, performance-assessment logic switches the control system from search to control mode (the system is in control mode most of the time), at which point adaptation or retraining of the predictive networks may be possible, using some of the system's computer resources. Adaptation is a fine tuning of network coefficient values, keeping the connectivity structure fixed, perhaps using a gradient or guided random search algorithm. Retraining completely restructures the networks and recalculates the coefficients to modify the ways in which network input variables interact within the networks themselves. Adaptation alone is usually sufficient, adjusting the coefficients in a background mode of control computer utilization—that is, with a lower priority than the control function, carried out only when the computer is briefly idle. Sometimes, adaptation is needed so infrequently that it can be performed entirely offline at, say, monthly intervals.

Sensor verification logic—an important part of the learning network system—establishes that each set of process measurements submitted to the networks is reasonably similar to the data patterns with which the networks were trained. If this similarity is not present, the process or sensor may be malfunctioning, or perhaps the system requires further training. Programs have been developed that perform the similarity test automatically and deal with each possibility that can arise; these programs use a data-clustering algorithm to synthesize the test.

Learning Network Example

Control of runout table cooling sprays in a hot-strip steel-finishing mill has been achieved, although this is a difficult industrial process for which to build a mathematical model—difficult because it has many process variables, strongly nonlinear interactions occur between some of these variables, and the response to a change in any input variable is delayed by a length of time related to the velocity of the steel strip moving through the mill and the physical length of the mill itself. Yet modeling such a process is extremely important, because it offers the best way to reduce or eliminate substantial economic losses suffered when either manual or conventional automatic controls are applied.

Learning networks provide a new method of modeling this process. They provide accurate, readily computable relationships with which—as the values of input and control variables are changed—outputs can be predicted. Because they take time delays into account, they are not sluggish as are linear controllers for transport delay plants, and they therefore avoid production of much off-specification material resulting from the large amount of time that such controllers require to reach their final states after input changes. Also, unlike some linear controllers that are too closely coupled, they do not become unstable.

Since it does not require that major nonlinear interactions be precisely known in advance, the learning network method is more successful than previously applied linear regression techniques. Also, unlike classical regression techniques, it is unlikely to go astray with real process data after having been made to work well with a different set of test data—because overfitting is avoided.

A typical hot-strip finishing mill consists of six roll stands, a runout table several hundred feet long with water sprays both above and below it, and a coiling device at its end. A steel bar about 1 in. thick and heated to approximately 1900°F enters the first roll stand, which squeezes it into a somewhat thinner and wider bar that is moving substantially faster than when it entered the stand (part of the steel displaced by squeezing goes sideways, part of it moves forward to contribute to the velocity of the bar). The other roll

stands repeat the process in succession, so that the output strip is perhaps 0.1 in. thick, cooled to 1600°F (but still red-hot) and moving at up to 2000 ft/min. As it moves down the runout table, the strip must be cooled by the sprays to a specific temperature before it is coiled.

For illustrative purposes, rather than showing how the method may be applied to an entire rolling mill, we discuss here only its application to control of pressurized water sprays used to cool the hot steel strip after it has passed the last roll stand. (These sprays prepare the strip for coiling.) A typical mill contains many such sprays, typically arranged in about 15 discretely controllable spray banks; they affect several parameters in the strip being produced, which can take several seconds to pass from the last roll stand to the coiling device.

In this example, the model, which is programmed on an IBM 1800 process control computer, predicts the number of sprays required to achieve the desired coiling temperature. It has seven inputs: coiling and finishing temperatures; the strip's speed, thickness, width, and hardness; and spray pressure. (Finishing temperature is at the last roll stand; the difference between the two temperatures represents the heat that must be removed by the sprays.) Outputs are number and configuration of the sprays, chosen from eight above the table and seven underneath.

Reaction of the bar to the successive rolling steps is initially assumed and the sprays are preset when the bar enters the first roll stand, according to a prediction based on this assumption. As the partially rolled strip emerges from the fourth stand, the predicted sprays are actually turned on; they take a few seconds to build up to their full volume, while the strip traverses the last two stands. When the strip emerges from the sixth and last stand, the prediction is made again on the basis of actual measured temperature and speed of the strip. If the assumed behavior of the strip was valid, the original prediction would have been correct and the proper sprays would now be operating; but if the measured variables depart in any way from their assumed values, the number of sprays is modified accordingly.

The model also predicts the temperature of the cooled strip as it begins coiling. If the actual temperature at this point differs markedly from that predicted, the model may need modification or further training, or the process sensors or actuators may not be working properly. In practice, 98% of a sample run of 612 coils fell within an acceptable tolerance of ±50°F, corresponding to a prediction error within ±2 sprays. Furthermore, 93.5% of the coils were within ±1 spray of the ideal setting—substantially better than the performance of conventional controllers, especially when product specifications are frequently changed. (The learning network model that produced this performance was synthesized from building blocks according to the procedure outlined earlier).

Other areas where learning networks have been applied or are currently being developed for a variety of CAM processes include inference of surface-finish roughness in machining, ultrasonic nondestructive testing, fermentation process modeling, crystallization process modeling, and modeling of casting quality in aluminum die-casting.

Conclusion

Results for control of the cooling sprays on the runout table of a hot-strip steel-finishing mill verify the utility of the learning network approach, even with conventional sensors. Since networks can be trained to make associations that ordinary modeling procedures cannot make, nonconventional sensors might be useful or even essential in other applications, demonstrating the power of the learning network approach even more dramatically.

Software realizations of learning networks provide adequate computing speed in many situations. For future applications requiring faster inferences or predictions, large-scale integrated microcircuits are being developed. These will lead to very flexible and powerful peripheral devices for the central processors to which they are attached.

Although CAM applications of learning networks have been emphasized here, CAD applications are also receiving attention. Whereas in CAM the network is trained to a high degree of accuracy before being used online, in CAD it learns with each successive design experiment, and with search logic it performs as a designer in seeking out the best design.

Bibliography

R. L. Barron, "Theory and Application of Cybernetic Systems: An Overview," *Proceedings 1974 IEEE National Aerospace and Electronics Conference*, pp 107-118

L. O. Gilstrap, Jr, "Keys to Developing Machines with High-Level Artificial Intelligence," *1971 American Society of Mechanical Engineers Design Engineering Conference and Show*, Paper 71-DE-21

D. Hampel and R. L. Barron, "Application Trends for Electronically Programmable Arrays," *Proceedings 1975 National Aerospace and Electronics Conference*, pp 470-478

D. Hampel, R. L. Barron, and D. Cleveland, "Design and Application of LSI Arrays," *Proceedings 1975 National Computer Conference*, pp 867-876

D. Hampel, R. W. Blasco, and D. Cleveland, "Electronically Programmable LSI Arrays," *Proceedings 1974 IEEE National Aerospace and Electronics Conference*, pp 134-141

A. N. Mucciardi, "Elements of Learning Control Systems with Applications to Industrial Processes," *Proceedings 1972 IEEE Conference on Decision and Control and 11th Symposium on Adaptive Processes*, pp 320-325

A. N. Mucciardi and E. E. Gose, "An Automatic Clustering Algorithm and Its Properties in High-Dimensional Spaces," *IEEE Transactions on Systems, Man, and Cybernetics*, 1972, pp 247-254

Roger L. Barron is president, chairman, and senior research scientist at Adaptronics, where he is active in R&D projects ranging from self-organizing control systems to process modeling/control. He holds a BSE degree from Princeton University and an SM degree from Massachusetts Institute of Technology.

APPENDIX C

ADAPTRONICS, INC. TECHNICAL NOTE

A NETWORK/CLUSTER CLASSIFIER

September 1974

by
D. Cleveland

Adaptronics, Inc.
Westgate Research Park
7700 Old Springhouse Road
McLean, Virginia 22101

INTRODUCTION

Mucciardi [1] has developed several procedures for augmenting cluster classifiers with trainable, polynomial networks. The networks transform the x-input data into a y-domain where the classes are more easily and accurately clustered. The classifier proposed here employs the same concept of transforming the x data into a "more clusterable" y domain. With the Mucciardi approach, at least one network is required for each class, so in many class problems, the number of networks is large. An attempt is made here to reduce the number of nets required to perform an adequate transformation, so classification problems involving many classes, e.g., twenty-five or more, become more tractable.

A NETWORK/CLUSTER CLASSIFIER

THE PURE CLUSTER CLASSIFIER

When the cluster algorithm is used by itself, i.e., with no network augmentation, as a classification tool it is "trained" as follows: The training data (the X vectors) for the K classes are separated according to class. For each class, the cluster algorithm constructs hyperellipses, defined by means $\bar{x}_{kn}$ and standard deviations σ_{kn}, of data which cluster closely. The subscript k denotes class and n denotes the component of the X vector. The locations and dimensions of these hyperellipses, along with the associated classes, comprise the information used by the classifier in the classification mode.

A block diagram of the cluster classifier is shown in Figure 1. Its operation, given an input vector X, is to generate a normalized distance measure indicating how far the vector X lies from each ellipse

$$d_{ki}^2 = \sum_{n=1}^{N} \frac{(x_n - \bar{x}_{kin})^2}{\bar{\sigma}_{ki}^2} \qquad (1)$$

The subscript i denotes the ith cluster within the kth class. (Table 1 gives a definition of the indices used in this note.)

The unnormalized probability of belonging to that hyperellipse is

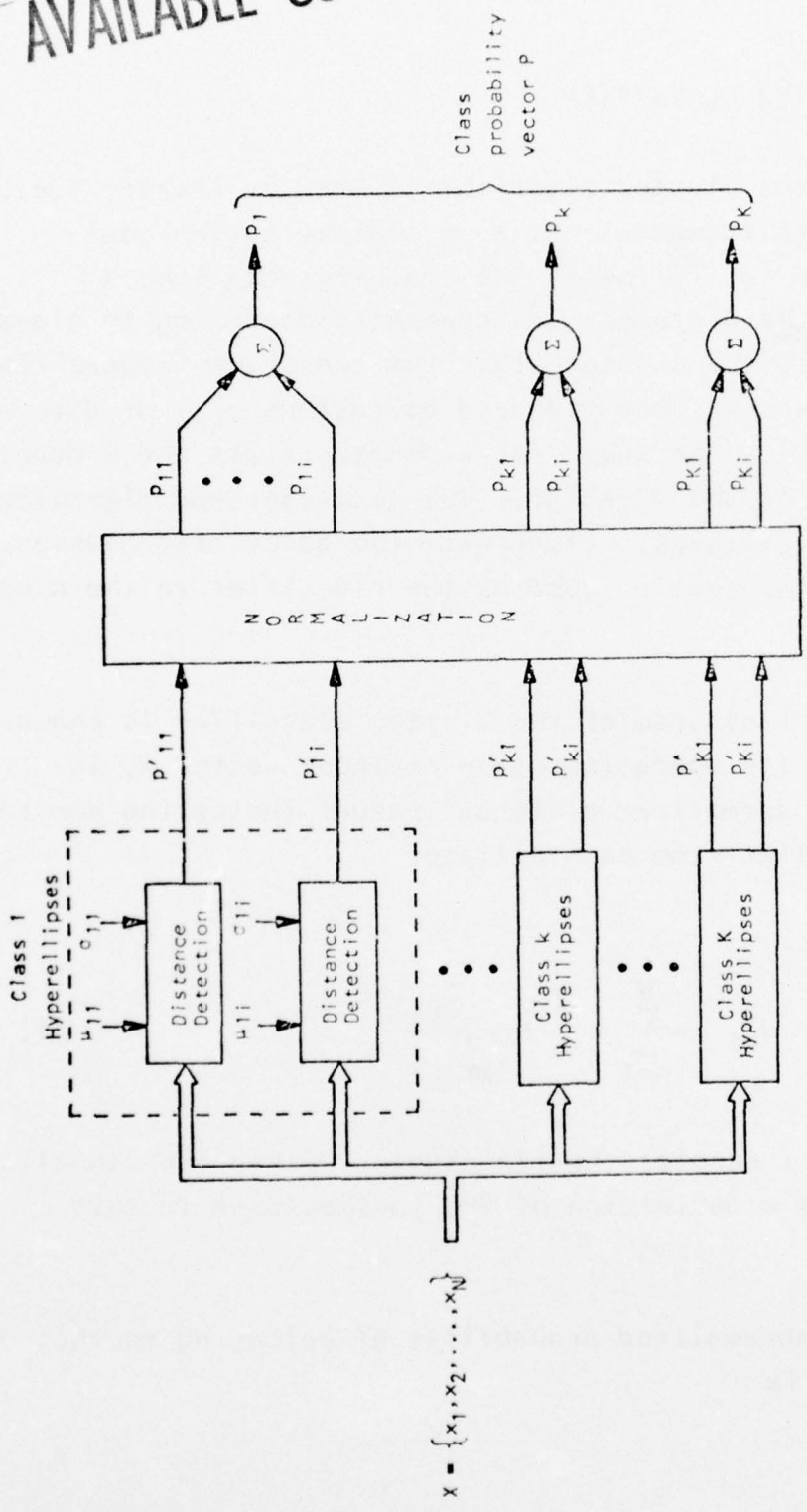


Figure 1: The Pure Cluster Classifier

Table 1: Definition of Indices

i	subscript indicating ellipse number
I	total number of ellipses per class
j	subscript indicating class number
K	total number of classes
k	subscript indicating class number
L	total components in the y vector
l	subscript indicating y vector component
M	total number of entries (in the training set) per class
m	subscript indicating entry number
N	total components in the x vector
n	subscript indicating x vector component

$$p'_{ki} = \exp(-d^2_{ki}) \tag{2}$$

and the normalized probability is

$$p_{ki} = \frac{p'_{ki}}{\sum_{k=1}^{K} \sum_{i=1}^{I} p'_{ki}} \tag{3}$$

The probability of belonging to class k is the sum of all the class k clusters.

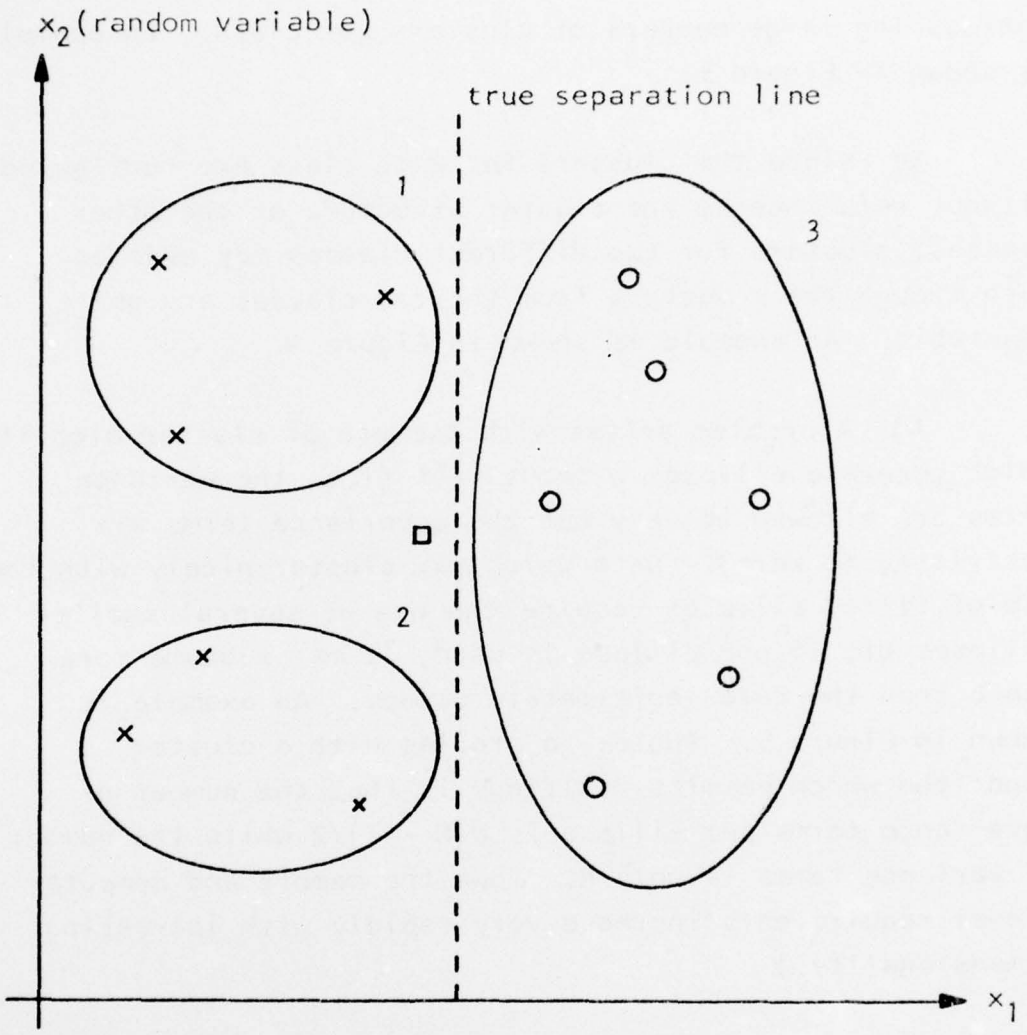
$$p_k = \sum_{i=1}^{I} p_{ki}$$

The class probabilities p_k form a class probability-state-vector (PSV).

Several problems may occur using the above classification routine:

1.) If any of the variables in the X vector is totally random, i.e., independent of the true classification information, several X data points (within a class) which would otherwise cluster, will be artificially separated in hyperspace due to variation on the random variable. Excess clusters are introduced to handle separated data, so more complex classification logic is required. Further, new data with values of the random variable which do not fall within the training set regions will not necessarily be classified properly. An example is shown in Figure 2.

2.) If various X data points (within a class) do not fall within elliptically shaped regions, several



□ new datum here is misclassified since it is closest to cluster 3

Figure 2: Artifical Data Separation

ellipses must be used to approximate the region. This increases the complexity of the cluster classifier, often introducing large numbers of clusters per class. An example is shown in Figure 3.

3) Since the clusters for each class are configured without reference to the cluster structure of the other classes, clusters for two different classes may overlap even though the x vectors from the two classes are quite separable. An example is shown in Figure 4.

4) A problem arises with the use of cluster algorithm which generate ellipses without tilt (i.e., the variance terms are allowed to vary but the covariance terms are restricted to zero). Data which may cluster nicely with the use of tilted ellipses require the use of several smaller ellipses or, if one ellipse is used, it may subsume more space than the data legitimately occupy. An example is shown in Figure 5. (Note: a problem with a cluster algorithm which permits "tilting" is that the number of covariance terms per ellipse is $N(N - 1)/2$ while the number of variance terms is only N. Thus the memory and computational requirements increase very rapidly with increasing dimensionality.)

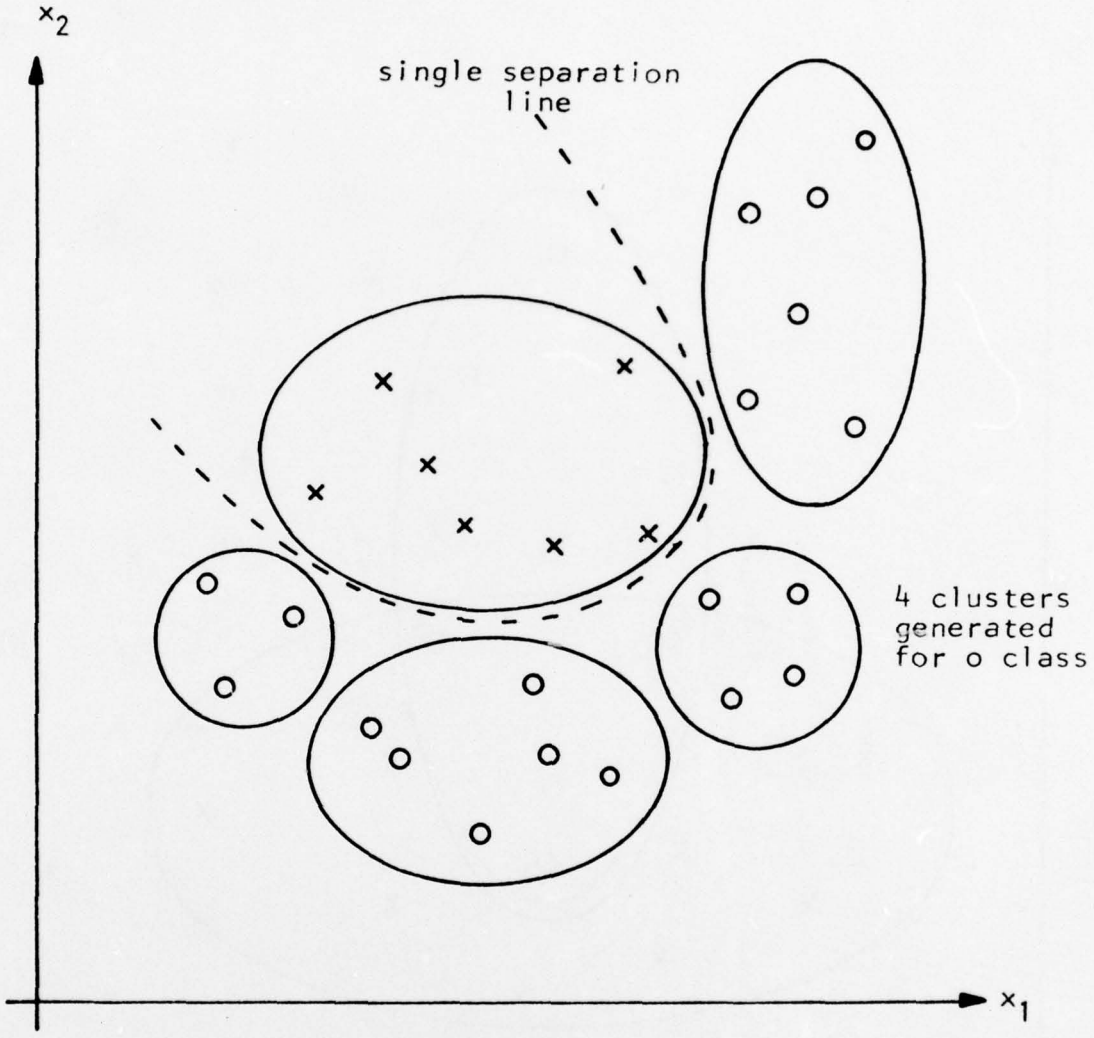


Figure 3: Excessive Clusters

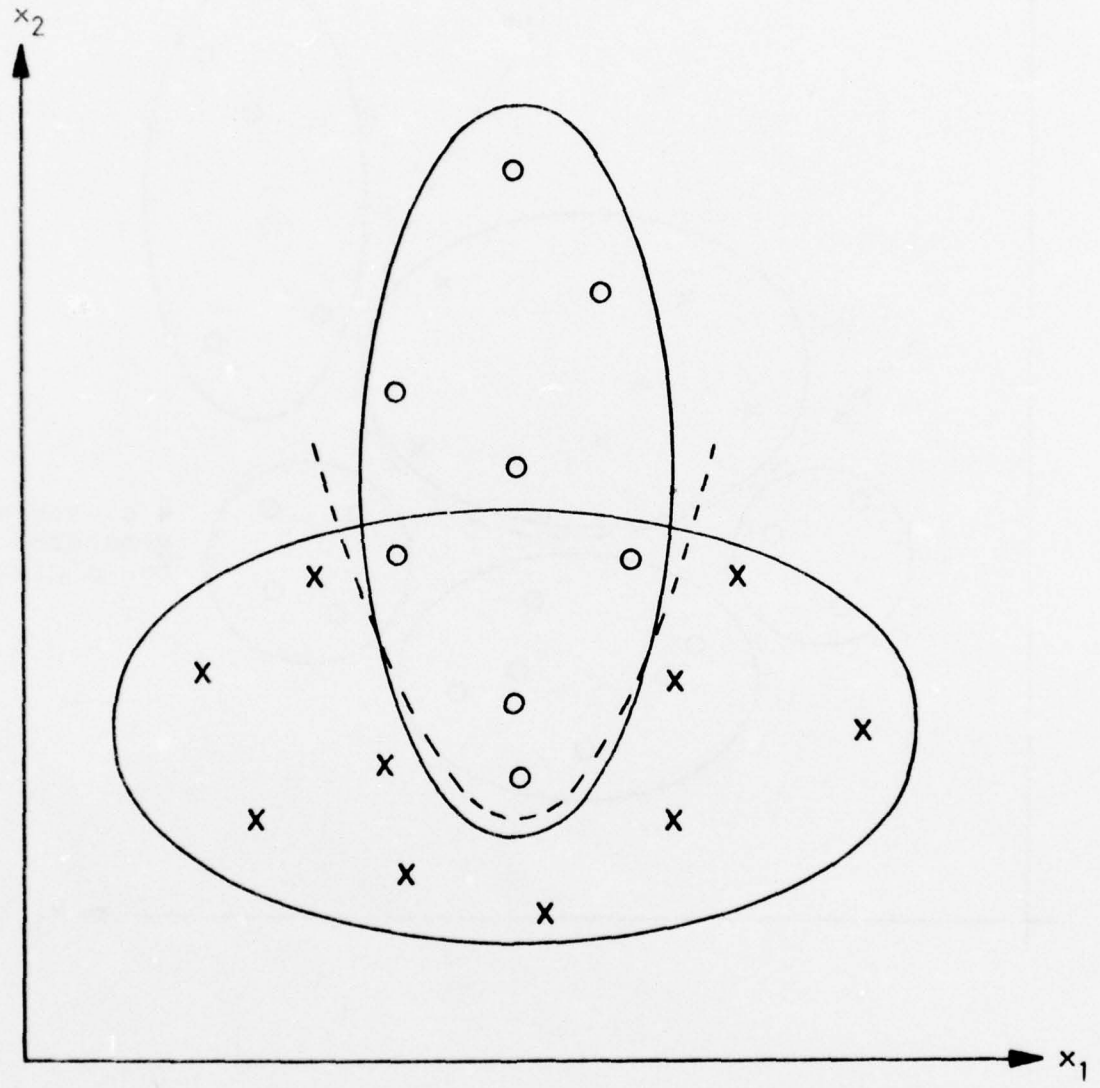
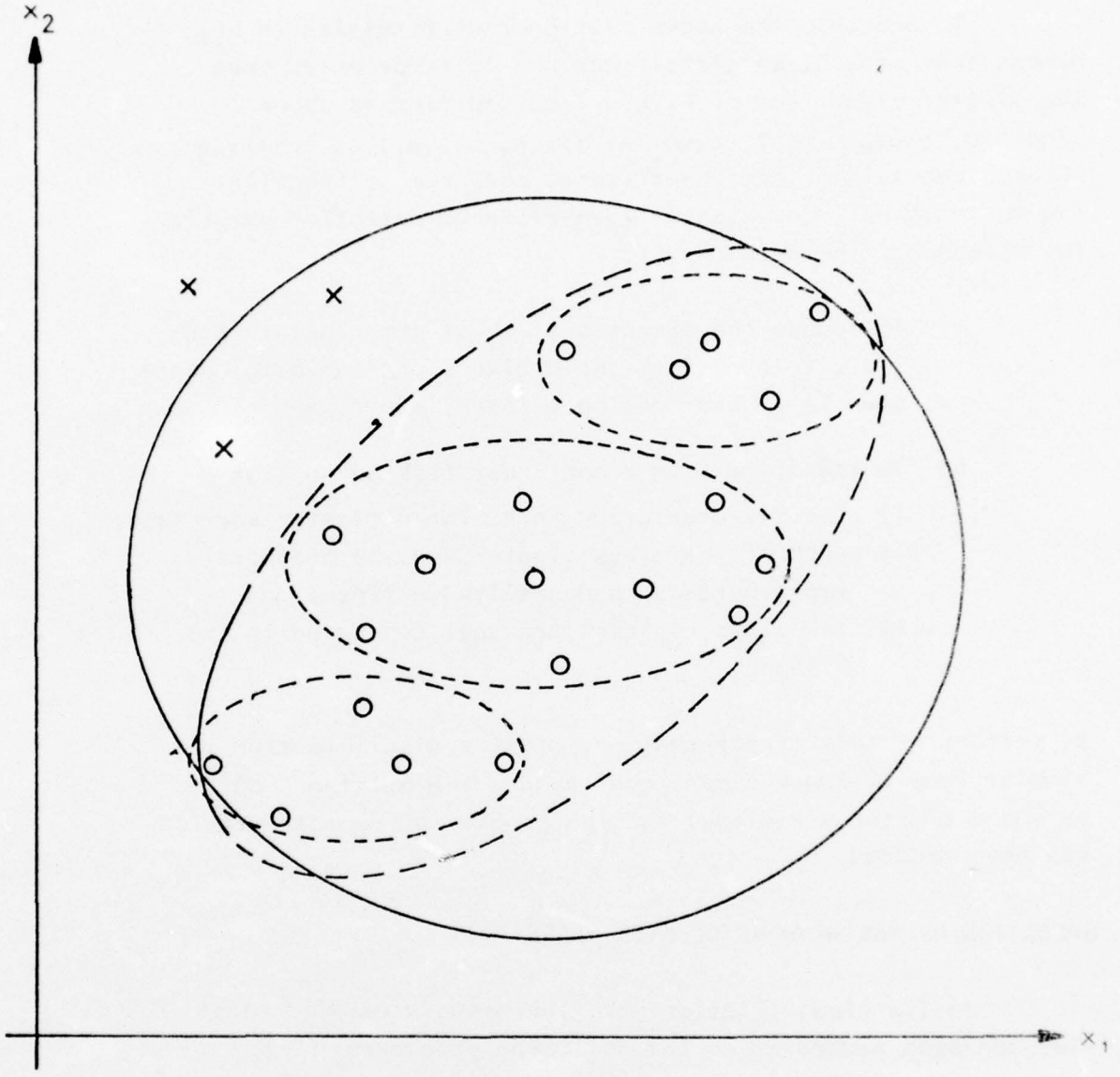


Figure 4: Cluster Overlap



——— single large "nontilted" ellipse

- - - - - - multiple small "nontilted" ellipses

— — — single large "tilted" ellipse

Figure 5: Tilted vs. Nontilted Ellipses

A NETWORK/CLUSTER CLASSIFIER

To overcome the above mentioned difficulties in high-dimensional, many-class classification, it is proposed that the cluster classifier of Figure 1 be modified as shown in Figure 6. a group of L networks, $(N_1, N_2, \ldots, N_L)$ is inserted between the X input and the distance measurement function, and there is only one hyperellipse-distance-detection per class. The purpose of the networks is:

a) To reduce the dimensionality of the cluster space, i.e., to transform the N dimensional X-vector space down to an L-dimensional ($L \ll N$) space, and

b) To transform X in a nonlinear fashion so that
   i) the Y vector group in a single cluster per class,
   ii) each of the class clusters may be adequately represented by a non-tilted ellipse, and
   iii) the class clusters are well separated in the Y space.

By performing this transformation, cluster discrimination is simpler (due to lower dimensionality and the existance of only one cluster per class) and the problem of cluster overlap has been reduced.

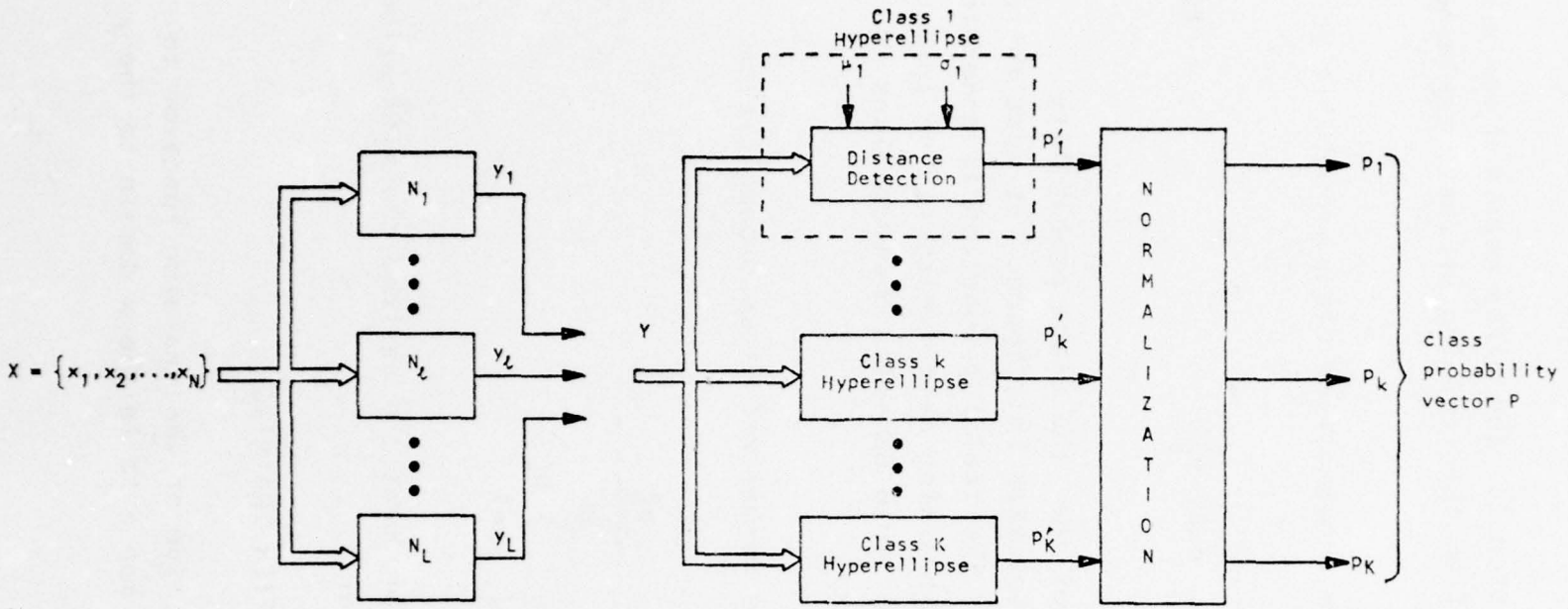
OPERATION OF THE NETWORK/CLUSTER CLASSIFIER

In its classification mode the network/cluster classifier operates according to the following procedure:

1) Compute the Y vector from the X vector with the use of network transformation. Each component y_ℓ is a polynomial function N_ℓ of the X vector.

2) Compute the normalized distance measure, in y space, to the K class clusters:

$$d_k^2 = \sum_{\ell=1}^{L} \frac{(\mu_{k\ell} - y_\ell)^2}{\sigma_{k\ell}^2} \qquad (5)$$



Note: N's are polynomial networks trained by a guided accelerated random search.

Figure 6: Network/Cluster Classifier

The y_ℓ data come from step 1. The class mean and variance data $\mu_{k\ell}$ and $\sigma^2_{k\ell}$ come from the training process, explained later.

3) Compute the unnormalized class probabilities:

$$p'_k = \exp(-d_k^2) \tag{6}$$

This computation assumes that the class probability - density-functions are Gaussian in y space. It must therefore be an objective of the training function to generate networks which map the training data (which may be irregularly spaced in the x domain) into Gaussian distributions in the y domain.

4) Compute the normalized class probabilities:

$$p_k = \frac{p'_k}{\sum_{k=1}^{K} p'_k} \tag{7}$$

The last step produces the desired classification probability-state-vector.

NETWORK/CLUSTER CLASSIFIER TRAINING

It is the objective of the training function to develop networks which map data in the x domain to the y domain where:

a) all the data from a given class k form a single Gaussian distribution in y space, and

b) the K class distributions in y space are sufficiently separated that inter-class ambiguities are minimized.

The classifier synthesis technique is to select a set of L polynomial networks with "sufficiently rich" generality that an "adequate" transformation from the x domain to the y domain can be achieved if the proper coefficients are selected. The coefficients are selected using a GARS algorithm. The performance measure for the search is mathematically complex, but it simply measures how well the networks are clustering the within-class data and separating the different-class clusters.

Within each class, the means and variances of the data in the y domain are

$$\mu_{k\ell} = \frac{1}{M} \sum_{m=1}^{M} y_{k\ell m} \tag{8}$$

and

$$\sigma^2_{k\ell} = \frac{1}{M-1} \sum_{m=1}^{n} (y_{k\ell m} - \mu_{k\ell})^2 \tag{9}$$

The "separation", which is a unitless measure, of class j from class k is given by the Mahalanobis D^2 metric: the squared differences of the means normalized by the "spreads" or variances of the class data around their respective means (summed over all L dimensions):

$$d_{jk}^2 = \sum_{\ell=1}^{L} \frac{(\mu_{j\ell} - \mu_{\ell k})^2}{\sigma_{j\ell}^2 + \sigma_{k\ell}^2} \qquad (10)$$

The aggregate separation S is given by the "parallel" combination of <u>all</u> the inter cluster separations:

$$S = \frac{K(K-1)}{2} \qquad \frac{1}{\sum_{k=1}^{K-1} \sum_{j=k+1}^{K} \frac{1}{d_{jk}^2}} \qquad (11)$$

The parallel combination of d^2 places maximum performance benefit on increasing the smallest inter-class separations, with minimum emphasis on increasing separations which are already large.

The performance assessment (PA) function for the GARS algorithm is to maximize the aggregate separation S.

Note that the PA function places no constraints on the location of clusters in y space. Any network solution for which the ratio of the cell sizes to the intercell distances (see Equation 10) is small is sufficient for unambigious classification.

Once a sufficient value for S has been obtained the values of cluster means and variances are obtained directly from Equations 8 and 9. These values are used when the classifier is used in the classification mode. A matrix representation of these data is shown in Figure 6a.

Also of interest is the separation matrix, shown in Figure 6b, which gives the interclass separation (from Equation 10) for all classes. This indicates which classes are likely to be confused and which are not.

a) y cluster data, means and variances:

y dimension

class		1	2		ℓ		L	
	1	μ_{11}	μ_{12}		$\mu_{1\ell}$		μ_{1L}	
	2	μ_{21}	μ_{22}		$\mu_{2\ell}$		μ_{2L}	
	k	μ_{k1}	μ_{k2}		$\mu_{k\ell}$		μ_{kL}	means
	K	μ_{K1}	μ_{K2}		$\mu_{K\ell}$		μ_{KL}	

class								
	1	σ^2_{11}	σ^2_{12}		$\sigma^2_{1\ell}$		σ^2_{1L}	
	2	σ^2_{21}	σ^2_{22}		$\sigma^2_{2\ell}$		σ^2_{2L}	
								variances
	k	σ^2_{k1}	σ^2_{k2}		$\sigma^2_{k\ell}$		σ^2_{kL}	
	K	$\sigma\sigma^2_{K1}$	σ^2_{K2}		$\sigma^2_{K\ell}$		σ^2_{KL}	

b) separation matrix:

	class	1	2		j		k
	1	0					
	2	d^2_{21}	0				
class	k	d^2_{k1}	d^2_{k2}		d^2_{kj}		
	K	d^2_{K1}	d^2_{K2}		d^2_{Kj}		0

Figure 6: Useful Information

REFERENCE

[1] Mucciardi, A. N., E. C. Orr, and R. Shankar, "Discrimination between Five Spoken Languages by Trainable Hypercomp™ Network Classifier," Adaptronics, Inc. Final Technical Report under Purchase Order No. 901705-0001-53-C28 for RCA Corporation, Camden, N. J., April 22, 1974.

APPENDIX D

ELECTRONICALLY PROGRAMMABLE LSI ARRAYS*

D. Hampel and R.W. Blasco
RCA Advanced Communications Laboratory

D. Cleveland
Adaptronics, Inc.

Abstract

A hardware approach has been developed for the realization of electronically programmable arrays for use in a variety of signal-processing functions. Such arrays comprise a network of elements whose input-output relationship, interconnectivity and constants may all be controlled or varied to solve particular problems.

Each element will be given the capability of evaluating any one of five polynomial expressions ranging from a linear combination of five variable inputs to the complete multinominal of two variables. The basic numerical format for all inputs, outputs and weights may be designated as fixed or floating point, with up to a 24-bit mantissa and 8-bit exponent.

It is shown how such an element can be multiplexed to simulate layers and then nets, or how a whole layer or net can be populated. The latter configurations provide higher throughput and reliability at the expense of arithmetic-circuit complexity. Factors of up to 1000 to 1 in speed improvement are realized with the reported hardware approach when compared to using general-purpose computers.

INTRODUCTION

It has been shown that arrays of calculating elements, providing various functions of their input variables, can be used as general-purpose signal processors.[1-8] Such elements could, in general, operate on binary, analog, or numerical inputs. Networks or arrays composed of elements in each of these domains have advantages in particular applications. The numerical processing element and its use in programmable arrays is the subject of this paper.

The major potentials of such arrays lie in their ability to perform reliable high-speed processing, and their ability to be used in adaptive control systems.[9,10]

The function repetoire of the basic element has been defined as a family containing both linear and non-linear multinomial expressions. The particular function of the element at a given time, its inter-connectivity within an array, and the coefficients or weights of its multinomial terms are all controllable. These features provide an array of such elements with a high degree of flexibility, and allows such an array to be alternately programmed to solve a large variety of problems.[13] Also, such an array can be "trained" in that it can rapidly accept different sets of weights and connection commands until it represents a transformation acceptable as a solution to a given problem. Although limited versions of such arrays have been built in hardware they have been simulated, for the most part, in software. This has confined the application of such arrays to off-line processing or experimental work.

With the objective of efficient array realization, an analysis of programmable array hardware requirements has been made, tradeoffs in its implementation have been completed, and an optimum architecture has been determined. Based on state-of-the-art LSI technology, projections as to array performance were derived. This derivation led, in turn, to the definition of a custom CMOS/SOS LSI circuit which would serve as a key ingredient in the processor of the element. This circuit and its use in programmable arrays will be described.

PROCESSING ELEMENT AND ARRAY STRUCTURE

A basic element is depicted in Fig. 1(a); the figure shows its major control and input and output signals. Such elements are, in general, structured in mult-layered nets, as shown in Fig. 1(b). The interconnection control, which is part of each element, but which appears functionally between successive layers of elements, has inputs which can route the output signals of

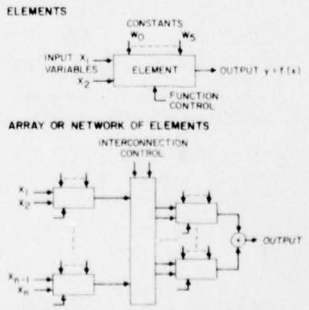


Fig. 1 - Programmable-function array structure.

ACKNOWLEDGMENTS (Page 8)

any one element to the desired input of an element in the next layer. Each element and interconnect switch in the array has sufficient memory for storing the function type it is to compute, the necessary weights or coefficients of its function, and the interconnect data. Hence, this programmable array can be considered as a distributed logic-memory processor capable of rapid reconfiguration and calculation of complex transformations.

Before describing the element requirements and architecture, the three possible configurations of programmable arrays will be explained; these configurations are shown in Fig. 2. In Fig. 2(a), a net of dimension j x k is shown fully populated, with each element containing an m-bit store for necessary function and control. This configuration can process (or transform) j/2 inputs at a time and can be operated in a pipeline mode so that k sets of data are simultaneously operated upon in different layers.

In Fig. 2(b), a single layer of elements can be multiplexed to simulate a whole net. The memory of each element must now have km bits to provide the necessary control as the processor of the element acts, in turn, to realize each of the k layers. For any single problem, the speed is the same as in Fig. 2(a), but pipeling can not be done. In Fig. 2(c) a single processing element with jkm bits of storage can be used to process inputs within a layer and then successive layers. Provisions must also be made for storage of intermediate results in Figs. 2(b) and (c); approach (c) will have 1/j the speed of (b).

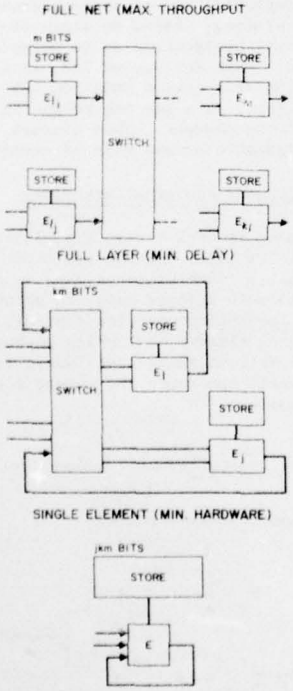


Fig. 2 - Possible array configurations.

It is envisioned that arrays of up to 256 elements (or equivalent) could provide the processing capacity for a vast majority of applications. Hence, the design, using either the configuration in Fig. 2(b) or (c), will have the capacity for memory expansion to simulate nets of up to 256 elements. These arrays will, in general, be loaded and controlled by a computer, but may have input/output signal interfaces as well, as shown in Fig. 3. As such, the programmable arrays can be considered as firmware, reconfigurable upon command, for high-speed processing.

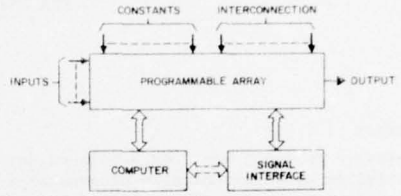


Fig. 3 - Programmable array environment.

ELEMENT DEFINITION AND DESCRIPTION

The basic numerical processing element of the programmable array will have the capability to perform the following functions:

P1 $y = W_o + W_1 X_1$

P2 $y = W_o + W_1 X_1 + W_2 X_2 + W_3 X_1 X_2$

P3 $y = W_o + W_1 X_1 + W_2 X_2 + W_3 X_1 X_2 + W_4 X_1^2 + W_5 X_2^2$

P4 $y = W_o X_2 - X_1 X_2^2$

P5 $y = W_o + W_1 X_1 + W_2 X_2 + W_3 X_3 + W_4 X_4 + W_5 X_5$

Functions P1, P2, and P3 are useful for general-purpose linear and non-linear hyper-surface calculations or transformations. P4 was provided for realizing division by a recursion formula. P5, a linear combination of five variables, is ideally suited for digital filters and linear transforms, such as the FFT. Each of the five functions could be realized by specifying only one element. For example, P1 could be used twice to realize P2, etc. However, substantially greater speed and efficiency is achieved by providing a micro-programmed control to optimally evaluate each function.

The element will be given the capability of operating on 32-bit floating-point values with a 24-bit mantissa and 8-bit exponent. It will also be capable of operating on fixed-point values of up to 24-bits with increased speed and lower package count.

The basic element is shown in Fig. 4; it consists of an arithmetic processor, an element controller, and random-access memories (RAM's).

The arithmetic processor performs additions, subtractions, and multiplications of the input variables and weights to compute the various polynomials. These operations may be either fixed or floating point, depending on the requirements of the array to be synthesized. Use of an iteration

scheme permits division to be simulated with several elements.

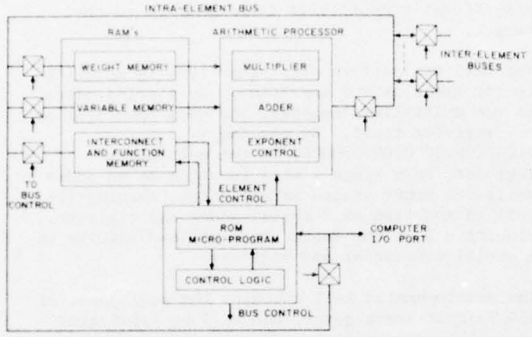


Fig. 4 - Basic element.

The element controller consists of a read-only memory (ROM) containing micro-instructions for implementation of the desired polynomial repertoire and associated logic to translate these micro-instructions into control signals for the arithmetic processor and address information for the RAM's.

The RAM's contain the polynomial function selection and element interconnection information, polynomial weights, and input/output variable storage for the element. Element interconnection is accomplished by specifying the RAM address of each required input variable. Output variables are stored in sequential RAM locations. If more than one element is used to simulate an array, the input variables might be stored in the RAM's associated with another element. In this case, one or more inter-element buses are provided to exchange data between elements. Since any element can access any previously generated output in this manner, complete interconnection flexibility is accomplished. An intra-element bus provides flexible data routing within the element. In operation, a computer "programs" each element by loading the proper polynomial select codes, input addresses, and polynomial weights into the RAM's via the intra-element bus. The computer then loads initial input variables into the variable memory via the intra-element bus, and provides an "execute" signal to the element controller.

Each element controller will sequentially step through the previously trained portions of its associated memory, performing the desired element operations and storing the results in the variable memory. This execute phase is complete when the elements detect a polynomial select code equivalent to a halt instruction. At this point the element controllers will output a "ready" signal to the computer. Upon receipt of the ready signal, the computer retrieves the output data by providing output addresses directly to the element controllers and receiving the output data via the intra-element buses.

DESIGN APPROACHES

Given the preceding operational constraints, what is the best technology and architecture to provide a good balance between speed and complexity or cost? The major hardware consideration impacting these criteria is the multiplier implementation. Two major organizational types of elements were investigated, one using a high-speed parallel multiplier (with and without pipelining) and the other a high-speed serial/parallel multiplier. Estimates as to total chip count and speed were made for each approach and for variations within each approach. Available LSI and MSI IC's are considered along with key LSI multiplier chips which would have to be developed for the overall element realization.

The necessary IC's to make the desired programmable arrays a viable processing scheme can be realized in a variety of emerging technologies. Very-high-speed LSI multipliers have been developed or are in development in both bipolar and CMOS/SOS form.[11,12] The choice of IC technology for each section of the element is dictated by availability and performance of existing circuits (particularly RAM's), as well as the best realization of any required new LSI development. These factors led to the definition of a CMOS/SOS serial/parallel multiplier chip compatible in speed and logic with associated control and memory. The serial nature of the design capitalizes on the "on-chip advantages" of SOS, and the high packing density of CMOS/SOS results in significant package-count savings. The parallel multiplier approach was based on an expandable 8 X 8 CMOS/SOS array. Alternate parallel-multiplier approaches were considered too expensive or not a good match with the rest of the element. Although there are faster multipliers, utilization of their speed to achieve substantially higher element throughput would require more elaborate control logic and RAM's.

For highest speed, a 24 X 24 bit parallel array multiplier, capable of pipeline (alternately) referred to as re-clocked or staged) operation can be used. The developmental 8 X 8 CMOS/SOS multiplier with a 100 nanosecond response time can optimally form the building block of such a multiplier as well as alternate approaches.[11,12]

A 2-stage pipelined multiplier appears optimum for that unit, with intermediate storage provided by available CMOS registers. The effective utilization of such a multiplier depends on a considerable number of gates for bus switching for its access from the RAM's and for scaling for floating-point justification.

The serial/parallel multiplier approach, basically a 1 X 24 accumulator, overcomes these disadvantages and also allows for the realization of elements over a relatively large range of performance factors by means of multiplier duplication. On the other hand, the parallel-array multiplier locks the design into relatively high speed and cost.

This serial/parallel multiplier approach will be described, and performance factors will be summarized and compared to the parallel multiplier.

LSI SERIAL MULTIPLIER CELL

The LSI serial multiplier cell currently under development is shown in Fig. 5. The cell shown calculates terms of the form ax + b, where a is the multiplicand, x is the multiplier, and b is the addend. The cell shown can handle addends and multiplicands in two's-complemented form of any desired length, and a multiplier in sign-magnitude form containing a sign bit and up to 23 significant bits. Cells may be cascaded to form higher-order terms or to handle larger multipliers.

Multiplication is accomplished in the 23 adder/latch stages by successively adding the contents of the multiplicand. The outputs of stages 7, 15, and 23 are brought out so that the cell may be efficiently used with 8, 16, or 24-bit multipliers (the first bit is the sign). The output of stage 1 is used to gain immediate access to the product sign bit to facilitate complementing operations on the product.

The multiplier register is divided into three serial-in/parallel-out registers to provide a good compromise between multiplier input leads and the time required to load the registers.

The two's complementer will complement the multiplicand if the multiplier sign bit is a 1. This operation results in a product which is in two's complemented form, as demonstrated in Table 1.

MULTIPLIER		MULTIPLICAND		TWO'S COMP MULTIPLICAND	CORRECTED MULTIPLICAND		PRODUCT	
SIGN	MAGNITUDE	SIGN	MAGNITUDE		SIGN	MAGNITUDE	SIGN[1]	MAGNITUDE[2]
+	X	+	Y	NO	+	Y	+	XY
+	X	−	2^N-Y	NO	−	2^N-Y	−	$X2^N-XY$
−	X	+	Y	YES	−	2^N-Y	−	$X2^N-XY$
−	X	−	2^N-Y	YES	+	$2^N-(2^N-Y)$ $+Y$	+	XY

1. THE PRODUCT SIGN IS EQUAL TO THE CORRECTED MULTIPLICAND SIGN.
2. $X2^N-XY$ IS A VALID TWO'S COMPLEMENT FORM IF THE PRODUCT IS TRUNCATED TO ELIMINATE OVERFLOW BITS BEYOND THE SIGN BIT POSITION.

Table I - Two's Complementer Operation

The Sign Hold lead in Fig. 5 inserts additional sign bits into the two's complement multiplicand input (equivalent to inserting zeros in front of a sign-magnitude number), and causes the most significant product bits to be shifted to the output.

The cell can perform other functions in addition to the basic ax + b operation. By placing zeros in the multiplier register, the cell functions as the register input. By placing 01000000000000000000000 in the multiplier register, only stage 1 will function as an adder while the other stages merely shift, causing the cell to function as a serial adder and register. Placing a 1 on the Negate Product lead results in a serial subtractor and register.

The developmental cell contains the equivalent of 450 2-input logic gates, and will be fabricated on a chip approximately 170 mils square by means of a double-epitaxial silicon-on-sapphire CMOS process. Expected maximum clock rate (based on computer simulations) is 25 MHz. The cell utilizes single-phase clocks, a single power supply, is static in operation, and may be mounted in a 16-pin package.

FIXED POINT ELEMENT

Fig. 6 shows ten multiplier cells arranged as a processor to implement the general second-order multinomial (P3). Cell 1 functions as a 23-stage delay register, cells 2 through 6 function as multipliers, cells 7 through 9 function as adder/multipliers, and cell 10 functions as an adder/register. All six terms of the multinomial are generated in parallel, and delays are matched so that a single clock is used for the entire processor. The ones complementer is used to convert the processor output to sign-magnitude form before storage in the X-Y variable RAM. Only the 23 most significant bits of the output are stored. (Truncation of the least significant bits and use of a ones complementer for the conversion to sign-magnitude form will result in plus or minus one LSB error in the stored output).

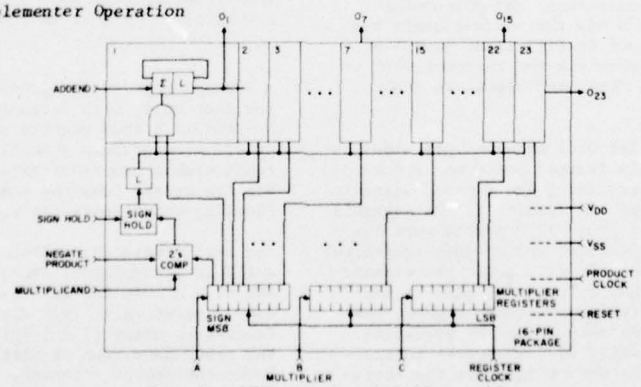


Fig. 5 - LSI serial-multiplier cell.

To implement a general-purpose arrays, gating must be added to route data from element to element The element control logic can then control the inter-cell data flow to form any given polynomial in the repertoire.

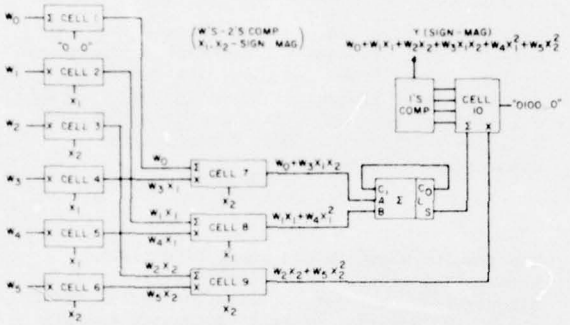


Fig. 6 - Serial fixed-point processor.

FLOATING-POINT ELEMENT

A floating-point arithmetic element is shown in Fig. 7. Floating-point arithmetic requires handling of an exponent, scaling of mantissas so that bits of equal significance are added together, and left-justification of the output mantissa (with corresponding correction of the output exponent) to retain the desired floating-point format.

Implementation of the floating-point element is shown in Fig. 8. The mantissa processor is similar to the processor of Fig. 6 except that provision is made for scaling of the terms before addition and for left-justification of the output.

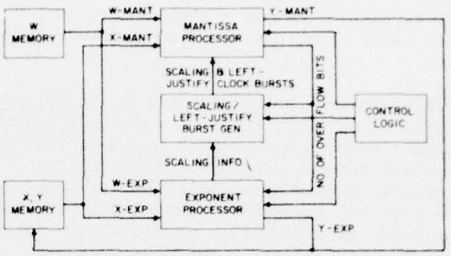


Fig. 7 - Serial floating-point processor.

The exponent processor performs the following operations:

a) Calculates the exponent for each term (e_i) and stores the largest e_i in the e_s memory.

b) Calculates ($e_s - e_i$) for each term and transmits this scaling information to the scalers.

c) Adds the "number of overflow bits" output of the overflow position detector to e_s to form the y-exponent output (corrected for left-justification).

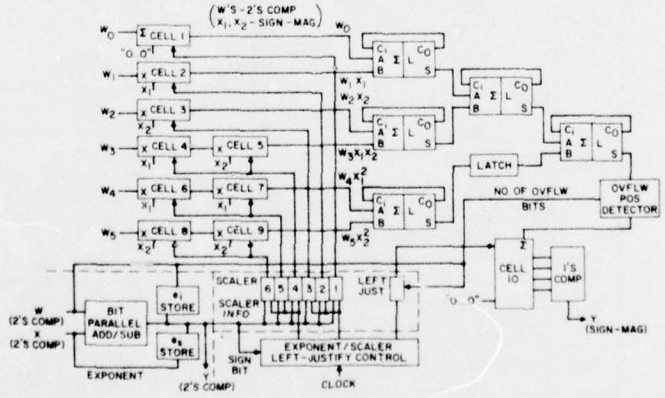


Fig. 8 - Floating-point processor.

The scalers delay the clocks to cells 1 through 3 to line up the mantissas of the first order terms with those of the second order terms. Each scaler then provides ($e_s - e_f$) additional clock pluses. Each additional pulse causes that term to be shifted right one position and decreases the significance of the bits in that term by a factor of two. In this way the terms are shifted relative to each other, so that bits of equal significance appear at the word-parallel adder inputs.

The overflow problem can be understood by assuming a binary point after the most significant bit of each mantissa. All mantissas are then equivalent to numbers between decimal 1 and decimal 2. For the general second order multinomial, it can be shown that the maximum is decimal 34. This range corresponds to a variation of six binary positions in the location of the output mantissa MSB.

The output mantissa is shifted until the least significant possible location of the MSB is located in cell 10 with the remaining possible "overflow" locations in the overflow position detector. This detector uses combinatorial logic to determine the MSB position and to generate a binary number representing this position. This binary number is used by the exponent processor to correct the output exponent, and by the left-justifier to generate the additional shift pulses to left-justify the output mantissa. The overflow circuit inserts the output sign bit adjacent to the output MSB to complete the formatting of the output.

The output mantissa is then ones complimented and stored in the X-Y variable memory along with the output exponent. As in the case of the fixed-point processor, the output mantissa will contain an error of ± 1 LSB.

COMPARISON OF APPROACHES

Detailed package count and speed estimates were made for the word-parallel/bit-serial processors of Figs. 6 and 8 and for alternate fixed- and floating-point processors by using a word-serial/bit-parallel approach with a 2-stage parallel-pipeline multiplier. These estimates are summarized in Table II.

The results shown in the table illustrate interesting trends in the word-parallel and bit-parallel approaches. With the serial multiplier, for example, circuit complexity varies linearly with the size of the mantissas while the product clock rate (based on a 25 MHz clock) does not depend directly on mantissa size. The complexity of a parallel multiplier, on the other hand, varies with the number of partial products which, in turn, vary as the square of the mantissa size. The speed of the parallel multiplier is limited by the propagation of carries when the partial products are added to form the total product, and thus also varies with the square of the mantissa size. (In the table, a 100-nanosecond cycle time is assumed for 16-bit mantissas while a 200-nanosecond cycle time is assumed for 24-bit mantissas).

PROCESSOR APPROACH			CUSTOM LSI MULTIPLIER PACKAGES	REQ'D[1] MSI/SSI ICS	REQ'D[2] ICS WITH SECOND CUSTOM LSI CHIP	SPEED(μs)					REMARKS
						P1	P2	P3	P4	P5	
FLOATING POINT	PARALLEL-PIPELINE MULTIPLIER	16 BITS MANT	4	190	73 LSI BUS SWITCH/SCALERS	0.5	0.8	1.2	0.7	0.9	–
		24 BITS MANT	9	250	114 LSI BUS SWITCH/SCALERS	1.0	1.6	2.4	1.4	1.8	–
	SERIAL MULTIPLIER	16 BITS MANT	10	80	10 LSI CONTROLLER SCALERS EXPONENT ACCUMULATOR	3.0	4.0	4.0	4.0	4.3	–
		24 BITS MANT	10	85	15 LSI CONTROLLER SCALERS EXPONENT ACCUMULATOR	4.0	5.2	5.2	5.2	5.2	–
FIXED POINT	PARALLEL-PIPELINE MULTIPLIER	16 BITS	4	110	68 LSI BUS SWITCH/SCALERS	0.4	0.7	1.1	0.6	0.8	USES 8X8 SOS MULTIPLIER
		24 BITS	9	150	84 LSI BUS SWITCH/SCALERS	0.8	1.4	2.2	1.2	1.6	COMPLEXITY ~ $\frac{1}{N^2}$
	SERIAL MULTIPLIER	16 BITS	10	27	2 ALL LSI	1.9	2.6	2.9	2.9	3.2	USES 24-BIT SOS SERIAL MULTIPLIER CELL
		24 BITS	10	30	3 ALL LSI	2.6	3.8	3.8	3.8	3.8	COMPLEXITY ~ $\frac{1}{N}$

1 EXCLUDING MEMORY

2 NO OF PACKAGES IN PROCESSOR EXCLUDING LSI MULTIPLIERS AND MEMORY

Table II - Package Count and Speed Estimates for Various Processors

In floating point operations, scaling of mantissas is accomplished in the bit-serial processors with a minimum of hardware at the expense of speed. Scaling in the bit-parallel processors conversely requires little time, but significant hardware is required.

The hardware constraints for the exponent processor for either approach tend to balance. The bit-serial mantissa processors (with longer processing times per term) require the scaling information for all terms to be available before any terms can be added, while the bit-parallel mantissa processors (handling the terms serially) only require the scaling information for one term at a time, although the processing time per term is shorter.

The associated MSI/SSI hardware is greater for the parallel-pipeline multiplier because of the need to control the flow of many parallel bits, while the need to serially shift data into the serial multiplier makes input/output and scaling occupy significant portions of the total processing time in the bit-serial processors.

The third column of Table II illustrates the reduction in associated MSI/SSI circuitry for each approach if an optimum second LSI chip-type were developed.

SUMMARY AND CONCLUSIONS

Elements for use in programmable arrays have been defined, and design approaches and performance indicated for alternate organizations. By giving an element the capability of evaluating one of five multinomial expressions with pre-set coefficients, upon command, and by providing means for interconnecting arrays of elements in various configurations, a number of signal processing functions can be realized. These include:

Classification and Control
Hyper-Surface Computation
Digital Filtering (Recursive, Transversal)
Transformations

The arrays can, in fact, be alternately and rapidly re-configured to do any of the above types of tasks. In that the transformation coefficients and multinomial degree and form can be readily controlled, an array can be trained to suit a particular situation.

Two types of design approaches were studied for realizing the hardware of a programmable element, one based on a parallel-pipeline multiplier and the other on a serial-type multiplier. In each case the multipliers would be realized by appropriate custom LSI, CMOS/SOS multipliers. Other circuit requirements are given, based both on a second LSI chip for control logic and on available IC packages. Although a parallel-pipeline multiplier can provide improved speeds of from 2-to-1 to 4-to-1 when compared to the serial-type multiplier, package counts were correspondingly greater. It was shown that the IC package totals for an element, based on all standard IC's besides the multiplier, were at least three times greater for the parallel-multiplier case. If a second custom LSI circuit is specified, the package savings with the serial multiplier would be at least 7 to 1 for floating-point and significantly greater for fixed-point elements. Considering that some or all of the speed differences between the two approaches can be compensated for, when required, by using a greater degree of parallelism in net organization (Fig. 2), the serial-multiplier approach was recommended for development. With a full 32-bit floating point (24-bit mantissa, 8-bit exponent) capability and a total of 10 of the custom LSI CMOS/SOS serial-type multiplier packages supported by 15 packages accomodating custom LSI control chips, a full six-term multinomial of 2-inputs can be evaluated in 5.2 microseconds. For 16-bit fixed point capability the control chips reduce to two, and the computation period reduces to 2.9 microseconds. The random-access memory to support each element will depend on its degree of multiplexing. For a fully populated array, up to 3 RAM packages would be required per element.

The advantages of such arrays include:

1. High-speed capability for general-purpose digital processing. The speed can be tailored by providing varying degrees of parallelism of the element in an array.

2. Capability for relieving software problems. The array can be considered as firmware once a set of coefficient and interconnect vectors have been found, or are known, for a particular application.

3. Capability for providing fault-tolerant computing. In general, alternate coefficient and interconnect vectors exist to realize the same transformation. Hence, if an element in a fully populated array is faulty, reprogramming of the array is possible by by-passing any faulty element(s). The degree to which this by-passing is done will, of course, depend upon the number of elements available and the complexity of the transformation.

ACKNOWLEDGMENTS

The guidance provided for this program by Mr. C. Gwinn of the U.S. Air Force Avionics Lab. is appreciated. Many helpful suggestions concerning the material in this paper and much of the background work leading to the reported concepts are due to the efforts and cooperation of Mr. R. Barron and Mr. D. Cleveland of Adaptronics, Inc. under sub-contract to RCA.

REFERENCES

1. Gilstrap, L.O., Jr., H.J. Cook and C.W. Armstrong, Study of Large Neuromime Networks, Adaptronics, Inc. Fin. Rept. Contract AF33(615)-5125, AF Avionics Lab., AFSC, AFAL-TR-67-316, AD#824 470, Dec. 1967.

2. Gilstrap, L.O., "Keys to Developing Machines with High-Level Artificial Intelligence", presented to ASME Design Conference, New York, April 22, 1971, (ASME paper #71-DE-21).

3. Cleveland, D., "Hardware Realization of Trainable Multivariable Nonlinear Transformations", presented to Computer Designers Conference, Anaheim, Calif. January 19-21, 1971.

4. Mucciardi, A.N. "Neuromime Nets as the Basis for the Predictive Component of Robot Brains", presented to ASC Fourth Annual International Symposium, Washington, D.C., October 8-9, 1970; to appear in Cybernetics and the Management of Ecological Systems, Robinson (Ed.), Spartan Books

5. Ivakhnenko, A.G. et al., "Group Handling of Data in Identification of the Static Characteristics of a Multi-External Plant", Soviet Automatic Control, 14, 2, 1969, pp. 30-37.

6. Ivakhnenko, A.G. and Y.V. Koppa, "Stochastic Algorithms and the Group Method of Data Handling in Prediction of Random Events," Soviet Automatic Control, 14, 3, 1969, pp. 20-32.

ACKNOWLEDGMENTS

The portions of this work relating specifically to development of certain Electronically Programmable LSI Arrays were supported by the U. S. Air Force Avionics Laboratory under Contract F33615-73-C-1089 to RCA with Adaptronics, Inc., subcontractor to RCA. The other portions of this work have been performed by Adaptronics, Inc. under the sponsorship of Armco Steel Corporation and the U. S. Air Force.

7. Ivakhnenko, A.G. and Y.V. Koppa, "Algorithm of the Group Method of Data Handling Using Linear Operators", Soviet Automatic Control, 14, 4, 1969, pp. 50-58

8. Ivakhnenko, A.G., "Polynomial Theory of Complex Systems", to appear in IEEE Trans. on Systems, Man, and Cybernetics.

9. Gilstrap, L.O., Jr., "An Adaptive Approach to Smoothing, Filtering and Prediction," Proc. of 1969 NAECON, pp. 275-280

10. Barron, R.L. and C.W. Gwinn, "Applications of Self-Organizing and Learning Control to Aeronautical and Industrial Systems", presented to ASME Design Engineering Conference, New York, April 19-22, 1971. (ASME paper #71-DE-22).

11. Hampel D., Micheel, L., Prost, K., "Threshold Logic Multiplier, NAECON 1973 Record, pp. 288-295.

12. McIver, G.W., Miller, R.W., O'Shaughnessy, T.G., "A Monolithic 16 X 16 Digital Multiplier," ISSCC Digest, Feb. 1974, pp. 54, 55.

13. Adaptronics, Inc., "Third Quarterly Progress Report for Armco Steel Corporation" (2nd Year), January 1, 1972.

APPENDIX E

Reprinted from —
AFIPS — Conference Proceedings
Volume 44

Montvale, N. J. 07645

Design and application of electronically programmable LSI arrays

by D. HAMPEL
RCA
Somerville, New Jersey
and
R. L. BARRON and D. CLEVELAND
Adaptronics, Inc.
McLean, Virginia

INTRODUCTION

It has been shown that arrays of calculating elements, providing various functions of their input variables, can be used as general-purpose signal processors.[1-9] Such elements could, in general, operate on binary, analog, or numerical inputs. Networks or arrays composed of elements in each of these domains have advantages in particular applications. The numerical processing element and its use in programmable arrays is the subject of this paper.

The major potentials of such arrays lie in their ability to perform reliable high-speed processing, and their ability to be used in adaptive control systems.[10-12]

The function repertoire of the basic element has been defined as a family containing both linear and nonlinear multinomial expressions. The particular function of the element at a given time, its inter-connectivity within an array, and the coefficients or weights of its multinomial terms are all controllable. These features provide an array of such elements with a high degree of flexibility, and allows such an array to be alternately programmed to solve a large variety of problems. Also, such an array can be "trained" in that it can rapidly accept different sets of weights and connection commands until it represents a transformation acceptable as a solution to a given problem. Although limited versions of such arrays have been built in hardware they have been simulated, for the most part, in software. This has often confined the application of such arrays to off-line processing or experimental work. With the objective of efficient array realization an analysis of programmable array hardware requirements has been made, tradeoffs in its implementation have been completed, and an optimum architecture has been determined. Based on state-of-the-art LSI technology, projections as to array performance were derived. This derivation led, in turn, to the definition of a custom CMOS/SOS LSI circuit which would serve as a key ingredient in the processor of the element. This circuit and its use in programmable arrays will be described. Examples of array applications in aerospace are presented.

PROCESSING ELEMENT AND ARRAY STRUCTURE

A basic element is depicted in Figure 1(a); the figure shows its major control and input and output signals. Such elements are, in general, structured in multilayered nets, as shown in Figure 1(b). The interconnection control, which is part of each element, but which appears functionally between successive layers of elements, has inputs which can route the output signals of any one element to the desired input of an element in the next layer. Each element and interconnect switch in the array has sufficient memory for storing the function type it is to compute, the necessary weights or coefficients of its function, and the interconnect data. Hence, this programmable array can be considered as a distributed logic-memory processor capable of rapid reconfiguration and calculation of complex transformations.

Before describing the element requirements and architecture, the three possible configurations of programmable arrays will be explained; these configurations are shown in Figure 2. In Figure 2(a), a net of dimension $j \times k$ is shown fully populated, with each element containing an m-bit store for necessary function and control. This configuration can process (or transform) $j/2$ inputs at a time and can be operated in a pipeline mode so that k sets of data are simultaneously operated upon in different layers.

In Figure 2(b), a single layer of elements can be multiplexed to simulate a whole net. The memory of each element must now have km bits to provide the necessary control as the processor of the element acts, in turn, to realize each of the k layers. For any single problem the speed is the same as in Figure 2(a), but pipelining cannot be done. In Figure 2(c) a single processing element with jkm bits of storage can be used to process inputs within a layer and then successive layers. Provisions must also be made for storage of intermediate results in Figures 2(b) and (c); approach (c) will have $1/j$ the speed of (b).

It is envisioned that arrays of up to 256 elements (or equivalent) could provide the processing capacity for a

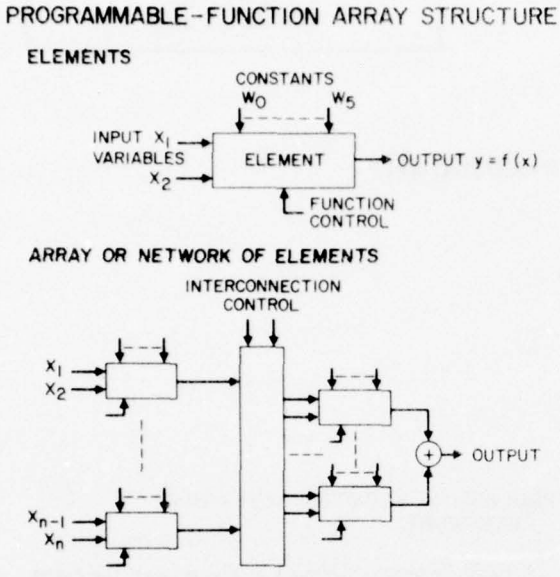


Figure 1—Programmable-function array structure

vast majority of applications. Hence, the design, using either the configuration in Figure 2(b) or (c), will have the capacity for memory expansion to simulate nets of up to 256 elements. These arrays will, in general, be loaded and controlled by a computer, but may have input/output signal interfaces as well, as shown in Figure 3. As such, the programmable arrays can be considered as firmware, reconfigurable upon command, for high-speed processing.

Element definition and description

The basic numerical processing element of the programmable array will have the capability to perform the following functions:

$P1\ y = W_0 + W_1 W_1$
$P2\ y = W_0 + W_1 X_1 + W_2 X_2 + W_3 X_1 X_2$
$P3\ y = W_0 + W_1 X_1 + W_2 X_2 + W_3 X_1 X_2 + W_4 X_1^2 + W_5 X_2^2$
$P4\ y = W_0 X_2 - X_1 X_2^2$
$P5\ y = W_0 + W_1 X_1 + W_2 X_2 + W_3 X_3 + W_4 X_4 + W_5 X_5$

Functions $P1$, $P2$, and $P3$ are useful for general-purpose linear and nonlinear hyper-surface calculations or transformations. $P4$ was provided for realizing division by a recursion formula. $P5$, a linear combination of five variables, is ideally suited for digital filters and linear transforms, such as the FFT. Each of the five functions could be realized by specifying only one element. For example, $P1$ could be used twice to realize $P2$, etc. However, substantially greater speed and efficiency is achieved by providing a micro-programmed control to optimally evaluate each function.

The element will be given the capability of operating on 32-bit floating-point values with a 24-bit mantissa and 8-bit exponent. It will also be capable of operating on fixed-point values of up to 24-bits with increased speed and lower package count.

The basic element is shown in Figure 4; it consists of an arithmetic processor, an element controller and random-access memories (RAM's).

The arithmetic processor performs additions, subtractions, and multiplications of the input variables and weights to compute the various polynomials. These operations may be either fixed or floating point, depending on the requirements of the array to be synthesized. Use of an iteration scheme permits division to be simulated with several elements.

The element controller consists of a read-only memory (ROM) containing microinstructions for implementation of the desired polynomial repertoire and associated logic to translate these microinstructions into control signals for the arithmetic processor and address information for the RAM's. The RAM's contain the polynomial function selection and elment interconnection information, poly-

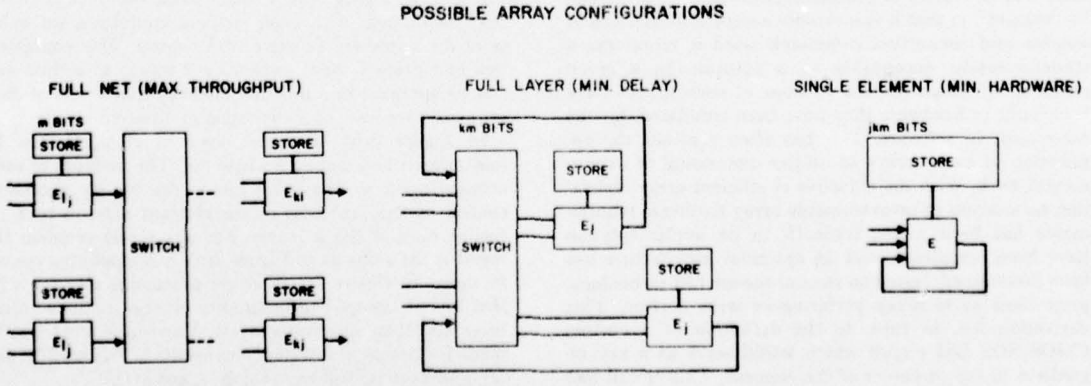


Figure 2—Possible array configurations

nomial weights, and input/output variable storage for the element. Element interconnection is accomplished by specifying the RAM address of each required input variable. Output variables are stored in sequential RAM locations. If more than one element is used to simulate an array, the input variables might be stored in the RAM's associated with another element. In this case, one or more inter-element buses are provided to exchange data between elements. Since any element can access any previously generated output in this manner, complete interconnection flexibility is accomplished. An intra-element bus provides flexible data routing within the element. In operation, a computer "programs" each element by loading the proper polynominal select codes, input addresses, and polynomial weights into the RAM's via the intra-element bus. The computer then loads initial input variables into the variable memory via the intra-element bus, and provides an "execute" signal to the element controller.

Each element controller will sequentially step through the previously trained portions of its associated memory, performing the desired element operations and storing the results in the variable memory. This execute phase is complete when the elements detect a polynomial select code equivalent to a halt instruction. At this point the element controllers will output a "ready" signal to the computer. Upon receipt of the ready signal, the computer retrieves the output data by providing output addresses directly to the element controllers and receiving the output data via the intra-element buses.

Design approaches

Given the preceding operational constraints, what is the best technology and architecture to provide a good balance between speed and complexity or cost? The major hardware consideration impacting these criteria is the multiplier implementation. Two major organization types of elements were investigated, one using a high-speed parallel multiplier (with and without pipelining) and the other a high-speed serial/parallel multiplier. Estimates as to total chip count and speed were made for each approach and for variations within each approach. Available

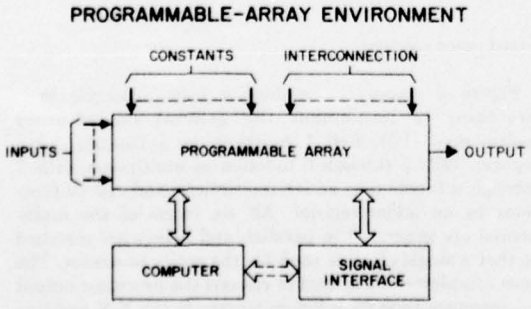


Figure 3—Programmable array environment

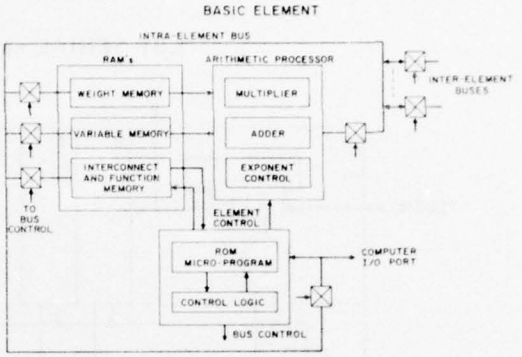


Figure 4—Basic element

LSI and MSI IC's are considered along with key LSI multiplier chips which would have to be developed for the overall element realization.

The necessary IC's to make the desired programmable arrays a viable processing scheme can be realized in a variety of emerging technologies. Very-high-speed LSI multipliers have been developed or are in development in both bipolar and CMOS/SOS form.[13,14] The choice of IC technology for each section of the element is dictated by availability and performance of existing circuits (particularly RAM's), as well as the best realization of any required new LSI development. These factors led to the definition of a CMOS/SOS serial/parallel multiplier chip compatible in speed and logic with associated control and memory. The serial nature of the design capitalizes on the "on-chip advantages" of SOS, and the high packing density of CMOS/SOS (compared to bipolar implementations) results in significant package-count saving. The parallel multiplier approach was based on an expandable 8×8 CMOS/SOS array. Alternate parallel-multiplier approaches were considered too expensive or not a good match with the rest of the element. Although there are faster multipliers, utilization of their speed to achieve substantially higher element throughput would require more elaborate control logic and RAM's. For highest speed, a 24×24 bit parallel array multiplier, capable of pipeline (alternately referred to as reclocked or staged) operation can be used. The developmental 8×8 CMOS/SOS multiplier with a 100 nanosecond response time can optimally form the building block of such a multiplier as well as alternate approaches.[13,14]

A 2-stage pipelined multiplier appears optimum for that unit, with intermediate storage provided by available CMOS registers. The effective utilization of such a multiplier depends on a considerable number of gates for bus switching for its access from the RAM's and for scaling for floating-point justification. The serial/parallel multiplier approach, basically a 1×24 accumulator allows for the realization of elements over a relatively large range of performance factors by means of multiplier duplication.

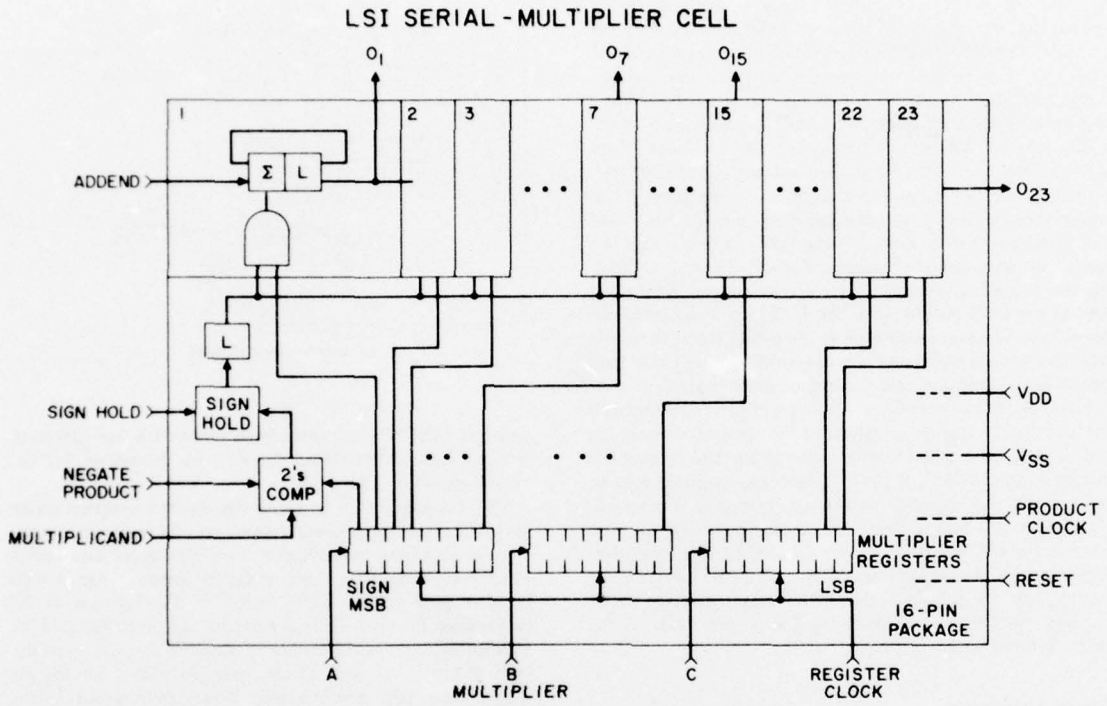


Figure 5—LSI serial-multiplier cell

On the other hand, the parallel-array multiplier locks the design into relatively high speed and cost.

This serial/parallel multiplier approach will be described, and performance factors will be summarized and compared to the parallel multiplier.

LSI serial multiplier cell

The LSI serial multiplier cell that has been developed is shown in Figure 5. The cell shown calculates terms of the form $ax + b$, where a is the multiplicand, x is the multiplier, and b is the addend. The cell shown can handle addends and multiplicands in two's-complemented form of any desired length, and a multiplier in sign-magnitude form containing a sign bit and up to 23 significant bits. Cells may be cascaded to form higher-order terms or to handle larger multipliers.

Multiplication is accomplished in the 23 adder/latch stages by successively adding the contents of the multiplicand. The outputs of stages 7, 15, and 23 are brought out so that the cell may be efficiently used with 8-, 16-, or 24-bit multipliers (the first bit is the sign). The output of stage 1 is used to gain immediate access to the product sign bit to facilitate complementing operations on the product.

The multiplier register is divided into three serial-in/parallel-out registers to provide a good compromise between multiplier input leads and the time required to load the registers.

The developmental cell contains the equivalent of 450 2-input logic gates, and has been fabricated on a chip approximately 170 mils square by means of a double-epitaxial silicon-on-sapphire CMOS process. The maximum clock rate is about 20 MHz. The cell utilizes single-phase clocks, a single power supply, is static in operation, and may be mounted in a 16-pin package.

Fixed point element

Figure 6 shows ten multiplier cells arranged as a processor to implement the general second-order multinomial (P3). Cell 1 functions as a 23-stage delay register, cells 2 through 6 function as multipliers, cells 7 through 9 function as adder/multipliers, and cell 10 functions as an adder/register. All six terms of the multinomial are generated in parallel, and delays are matched so that a single clock is used for the entire processor. The ones complementer is used to convert the processor output to sign-magnitude form before storage in the X-Y variable RAM. Only the 23 most significant bits of the output are stored. (Truncation of the least significant bit is used of

a ones complementer for the conversion to sign-magnitude form will result in plus or minus one LSB error in the stored output.)

To implement general-purpose arrays, gating must be added to route data from element to element. The element control logic can then control the intercell data flow to form any given polynomial in the repertoire.

Floating-point element

A floating-point arithmetic element can also be implemented. Floating-point arithmetic requires handling of an exponent, scaling of mantissas so that bits of equal significance are added together, and left-justification of the output mantissa (with corresponding correction of the output exponent) to retain the desired floating-point format.

TRAINING AND ADAPTATION OF MULTINOMIAL NETWORKS

We now consider the methods for training and adaptation of multinomial networks that are constructed from the elements described above.[12,15] This is done with special search algorithms, generally off-line, to specify network configurations. Applications of multinomial networks generally involve operations with sensor data that are obtained as a result of "observing" a physical object, process, or phenomenon. The classical approach to design of computer models for inferences and predictions from observations has been to determine all the relevant characteristics, deterministic and/or statistical, of the process being observed, and to use these measurements (and assumptions) in design calculations. Very often the structure of the model is presumed and the design takes the form of calculating the values of certain parameters. Even if the nature of the observed process changes, the structure of the model often does not change, but the parameter values are adjusted in response to measured changes in the inputs or in the outputs.

In many important applications, the inputs (i.e., the observables) are difficult to describe analytically. The best or even a good structure for the model cannot be determined *a priori*. In this case, it is desirable to have a model structure that can adjust to representative inputs. That is, the model is *trainable* both in its structure and in its parameter values.

It is therefore desired to implement a general (usually nonlinear) function of certain input variables which we can call observables. Since little may be known about the characteristics of the observables, the parameters of the network are not known *a priori*. The network will have to be trained with representative inputs. The questions are now:

1. How should the element parameters be adjusted?
2. How should the elements be interconnected and what should their complexity (i.e., number) be?

To make the ideas clear, suppose that the input consists of N observables, $x_1, x_2, \ldots, x_N$. Also suppose that the output is a scalar whose value may be considered as the estimate of some property of the input process. In general, y will be some nonlinear function of the x_i's as follows:

$$y = f(x_1, x_2, \ldots, x_N)$$

Polynomial (multinomial) approximation

Under fairly general conditions, a function of N variables may be expressed in an N-dimensional Maclaurin series as follows:

$$y = a_0 + \sum_{i=1} a_i x_i + \sum_{i=1} \sum_{j=1} a_{ij} x_i x_j + \sum_{i=1} \sum_{j=1} \sum_{k=1} a_{ijk} x_i x_j x_k + \ldots \quad (2)$$

In the most general case, the coefficients, $a_0, a_i, \ldots$, are functions of time, but for many cases of interest, the underlying characteristics of the x's do not depend on time and consequently the coefficients are constants.

Two questions which arise in the use of Equations (1) and (2) are:

1. What should the observables or measurements x_i be?
2. How many terms in Equation (2) will provide an acceptable approximation to the desired function, even though this (true) function is not known?

The answer to the first question is: All those that are *thought* to have a bearing on the desired output are used initially, and the ones which trial shows to be of little use are discarded. The second question is answered adaptively by using a trainable nonlinear network whose complexity determines the number of terms in Equation (2). The trainable network consists of interconnected elements, each of which implements a simple nonlinear function of

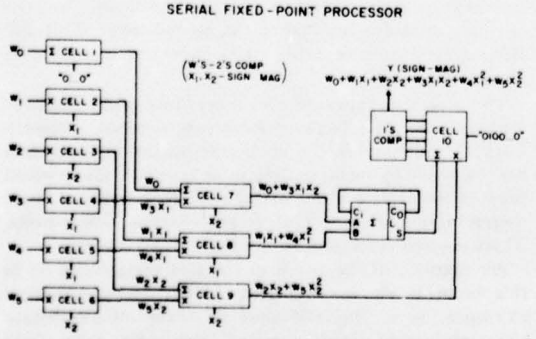


Figure 6—Serial fixed-point processor

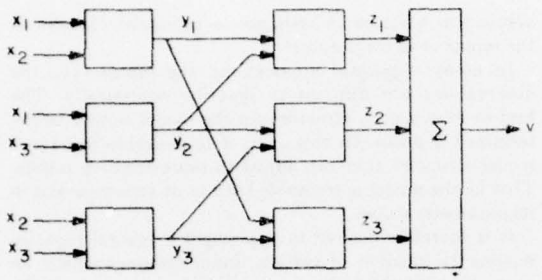


Figure 7—Two-layered network of elements

two inputs. The total network can be trained to provide an acceptable approximation to Equation (2).

A basic element of the learning network is the two-input, single-output device, illustrated in Figure 1 (and previously described) which implements the following function ($P3$) of its input x_1, x_2:

$$y = w_0 + w_1x_1 + w_2x_2 + w_3x_1x_2 + w_4x_1^2 + w_5x_2^2 \qquad (3)$$

Networks of the basic element

Suppose now we consider two layers of such elements as illustrated in Figure 7. It can be seen that each z_i contains pair-wise products up to degree four. Note that the first layer contains all possible pairs of three inputs x_1, x_2, x_3. To implement a general multinomial (i.e., a polynomial in many variables) expression, the number of elements in each layer would have to grow as one proceeds deeper into the network. However, it is found empirically that acceptable approximations are obtained without this growth; in fact, the number of elements in successive layers will soon (after, say, two or three layers) decrease, until only a few are left as inputs to the final network component (which is an adder).

The known data set

Now we turn to the matter of determining the coefficients of each network element and the number and interconnections of the elements.

These tasks are accomplished with a "known" data base; that is, a data base for which the values of the dependent variable are known. The steps involved are:

1. Optimizing the coefficients in each element of the first layer;
2. Selection of those elements whose output is acceptable (rejection of poor performers);
3. Repetition of the process for each layer; and
4. A global optimization (adaptation) of all coefficients in all layers based upon the final output.

The known data base is divided into three independent but statistically similar subsets:

1. Fitting subset
2. Selection subset
3. Evaluation subset

The fitting subset is used to determine the coefficients of the elements. The selection subset is used to reject the poor performers. The fitting and selection subsets are also used for the global optimization. The evaluation subset is used to estimate the overall performance. Since the evaluation subset was *not* used for network synthesis, the performance on this subset is an accurate estimate of the ability of the network to generalize to new, previously unseen data.

Training the network

The element coefficient determinations are based, in part, upon a least-square fit to a desired output. Other criteria are of course possible and are often used. Employing a least-squares criterion, the elements are first adjusted by a matrix algebraic procedure and then by a recursive search (i.e., optimization) procedure. An outline of the steps follows.

The fitting and selection subsets are used alternately in training each layer. The fitting subset is used first to establish the coefficients. The specific observables to be used initially have already been chosen. Let these be designated by $x_1, x_2, x_3, \ldots, x_N$. These are arranged in pairs, x_i, x_j; $i, j = 1, \ldots, N$. There are $N(N\text{-}1)/2$ such pairs. Thus, the first trial will require $N(N\text{-}1)/2$ trainable elements (such as that shown in Figure 1). A pair of observables is sent to each element. The coefficients of the element are determined using a recursive search procedure with a least-squares criterion. The procedure is repeated for each of the $N(N\text{-}1)/2$ elements.

Not all of the pairwise combinations are of significant aid in extracting the desired information. The selection process, is inserted into the first layer, resulting in $R(R\text{-}1)/2$ pairs of inputs into the $R(R\text{-}1)/2$ initial elements of the second layer. The coefficients of each element in the second layer are determined as in the first layer. Then the selection subset is applied to the second layer. This will eliminate the unacceptable pairs from the second layer inputs.

The process is repeated with succeeding layers until the error rate on the selection subset is minimized. Although further reductions in the error rate on the fitting subset are realizable by incorporating more layers, to do so would produce over-fitting of the fitting data. Eventually, a single output results from each of several disjoint subnetworks. These outputs are added to produce a single output.

An example of the result of the training process up to this point is shown in Figure 8. In this hypothetical example, it is implied that at least 30 candidate parameters were initially inserted into the first layer. Only a few survived, as indicated. The figure shows that pair

(x_1, x_{28}) interacts with pair (x_4, x_{30}). But pairs (x_5, x_8) and (x_{18}, x_{29}) do not interact with each other or with the other pairs. Thus, there are three disjoint subnetworks whose outputs are added to produce a single output.

There is a final step in the training process. This is a process of vernier adjustment, or "fine tuning" of the coefficients. If the need for this vernier adjustment arises, it is because the coefficients of each element have been adjusted in the absence of interactions with other elements following them in the network. The optimum coefficient values may be different when these interactions occur. The *fitting* and *selection* subsets are also used for this final adjustment process. The vernier adjustment is a global search and may use a random search technique to obtain the final values of the coefficients. The same global search adjustment may be used for subsequent adaptation of the network.

After the final adjustment of coefficients, the evaluation subset is used to estimate the performance of the entire network.

Avoidance of overfitting is a key aspect of the training of learning networks. Good functional approximations to the fitting data subsets are obtained that are also good approximations to the data in the separate selection subsets. This means that the networks are taught to generalize properly on their experience in fitting the points in the first subsets and that error rates in later uses will therefore be small. Without avoidance of overfitting, the networks would give deceptively small errors in approximating their first sets of data and then, in most cases, do poorly on subsequent new data. We have all seen or heard of empirical models that appeared to have much promise initially, but that produced unacceptable errors when presented with new data; in most cases, such behavior is the result of overfitting. By using three independent subsets of the available data, taking care that each is statistically representative of the whole data base, the problem of overfitting is virtually eliminated and good advance estimates of operational error levels of the models are obtained.

A corollary to the guarantee that models realized by learning networks are not overfitted is the fact that these models are smoothly-fitted functional approximations. From the mathematical standpoint, they are continuous and differentiable functions, and the derivatives of these functions are close approximations to the quantitative derivative behavior of the real processes that are modeled. For this reason, one may compute numerical partial derivatives of the form $\partial y/\partial x_i$. These derivatives reveal the quantitative sensitivity of the modeled variable (y)—and thus of the process that has been modeled—to small variations about specified values of the network input variables. Once a process is modeled, or a curve fit, by computer algorithm, the hardware elements (array) described previously can be programmed for high-speed real-time evaluation of new data.

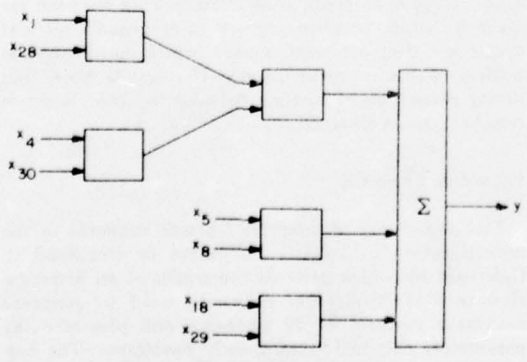


Figure 8—Illustrative learning network

AEROSPACE APPLICATIONS

Aerospace applications of the array/network concepts presented above include:

1. nondestructive inspection of structural parts
2. trajectory predictions
3. target signature classifications
4. radar refractive index corrections
5. detection of remote nuclear events
6. voice data processing
7. reconnaissance image processing
8. electronic warfare
9. avionics information systems

The status of work in several of these areas will now be summarized briefly.

Nondestructive inspection

One of the most representative applications to aerospace systems of the array/network concepts presented in this paper is that of nondestructive inspection of critical structural parts of aircrafts or missiles. The classical problem in inspection by such means as ultrasonic testing is that small defects such as fatigue-induced microcracks may be masked by reflecting surfaces such as fasteners and component surfaces. Although much theoretical work has been done to attempt to characterize the pulse echoes obtained from microcracks and other defects, no all embracing theoretical formulation exists at this time. The adaptive training of a multinomial network offers an attractive approach to the processing of pulse echoes in the complex signal environment typical of ultrasonic inspection applications. Data may first be gathered on test specimens having known properties and used to train the network in accordance with the procedures outlined above. Such work is progressing at the present time. It has already been demonstrated that the adaptive learning network can be trained to classify correctly flat-bottom hole defects in 7075-T6 Aluminum test blocks, using transducers of various diameters and differing band-pass

characteristics, centered in the neighborhood of five megahertz. As reported in Reference 16, in a total sample of 48 flat-bottom hole defects, 46 were correctly classified by this technique. An important aspect of the use of the multinomial network method is that it is not necessary to specify *a priori* the classifier input parameters that are most informative for a given application. In the case of the ultrasonic nondestructive inspection application, 96 candidate waveform parameters were considered during network training. Fifteen of these parameters were found to be relevant to the flat-bottom hole classification. These parameters are those that describe the overall shape and content (area) of certain parts of two waveforms computed from the pulse echo waveforms—the Power Spectrum and its log Fourier transform, the Cepstrum. Interestingly, maximum amplitude of the time waveform was not found to be a discriminating parameter when the transducer and/or transmission medium were subject to variation. The resultant network structure found for flat-bottom hole defect classification consists of 13 elements containing a total of 78 coefficients, and implementing an eighth degree function of the 15 input variables.

If application to the fatigue crack specimens is also successful, the adaptive learning network will provide an inspection tool of great value to the maintenance of aircraft and missiles in the operational inventory. The work to date has already demonstrated that the new procedures allow one to synthesize signal processors that deal with waveforms from the real environment and that learn which parameters of these waveforms are the most relevant, while combining these parameters into an adaptively trained signal classifier. Other applications have been suggested for the methodology in the related areas of acoustic emission testing, computer-aided manufacturing, optimization of metal removing processes, control of forging and casting, inference of material physical properties from microstructure data, and forecasting of maintenance necessitated by stress corrosion deterioration of flight vehicle structures. In each instance, the key point is that networks can learn to infer or predict from the natural data that are produced by the processes. These "natural" data may be records of process sounds, vibrations, deterioration due to corrosion, etc.—anything readily accessible for economical instrumentation or recordkeeping.

Trajectory predictions

Successful R&D has been conducted for more than a decade for application of multinomial networks to the prediction of aerospace vehicle trajectories. In summary, these investigations have established that the network methods are capable of inferring vehicle parameters, such as ballistic coefficients, quite accurately. Additionally, the networks make extremely fast, accurate predictions of object trajectories. These predictions are comparable in accuracies to the conventional procedures whereby equations of motion are integrated in serial computers, but are very much faster because of the computing speed of the parallel network structure.

Target signature classifications

Attention is being increasingly directed toward the automatic classification of target signatures from single or multiple sensors. An example of this work is that of classifying the sources of ground vibrations and/or acoustic emissions monitored by sensors in air drop ordnance. Very promising results are being obtained in such work.

Radar refractive index corrections

Reference 17 presents a new approach to the problem of computing height correction for aircraft or other objects tracked by surface radars. The basic procedure, when using the multinomial network, is to operate a cooperative aircraft that is equipped with an accurate radar altimeter in those regions of airspace for which the true altitudes of unknown targets are to be obtained. The cooperative target is tracked and an adaptive learning network is used to model the relationship between observed range, elevation angle and azimuth angle, and the independently measured height of the cooperative target. It is then possible to interrogate the network very quickly whenever the true height of an unknown target is to be computed. The inputs to the network become the apparent range elevation and azimuth of the unknown target, and the network output is the estimated true height of that target. The structure and coefficients of the network are adjusted adaptively as atmospheric conditions change. The reference presents a comparison between the accuracy of this approach and that of the conventional procedure. The height error, using the new approach, is approximately ⅓ of the error obtained with the prior state-of-the-art method.

Detection of remote nuclear events

Reference 18 presents results of work performed to assess the accuracy of an adaptive learning network classifier for discrimination between remote underground nuclear events and deep-core earthquakes (which masquerade as nuclear events in many cases). The results show that nearly perfect discrimination between the two classes of remote events is obtained.

Voice data processing

The application of adaptive learning networks to the identification of spoken languages is discussed in Reference 19, which presents the results of an investigation into multinominal networks used to generate nonlinear features of 29 phoneme and phoneme-like parameters obtained from speech waveforms. The net-

works are trained to discriminate between each pair of languages in the set of languages to be identified. The outputs of the networks are then input to a decision logic to identify which language is being spoken. In the case of five languages to be discriminated, this means that a group of nonlinear transformations is produced by 10 individual networks, each of which maps the 29-dimensional input space onto a component of a 10-dimensional output space. The structure of a typical trained network is shown in Figure 9. The results of the cited investigation show that significant improvement is obtained in the accuracy of language identification and in insensitivity to idiosyncracies of individual speakers.

SUMMARY AND CONCLUSIONS

Elements for use in programmable arrays have been defined, and design approaches and performance indicated for alternate organizations. By giving an element the capability of evaluating one of five multinomial expressions with pre-set coefficients, upon command, and by providing means for interconnecting arrays of elements in various configurations, a number of signal processing functions can be realized. These include:

Classification, Prediction and Control
Hypersurface Computation
Digital Filtering (Recursive, Transversal)
Transformations

The arrays can, in fact, be alternately and rapidly reconfigured to do any of the above types of tasks. In that the transformation coefficients and multinomial degree and form can be readily controlled, an array can be trained to suit a particular situation.

Two types of design approaches were studied for realizing the hardware of a programmable element, one based on a parallel-pipeline multiplier and the other on a serial-type multiplier. In each case the multipliers would be realized by appropriate custom LSI, CMOS/SOS multipliers.

Considering that some or all of the speed differences between the two approaches can be compensated for, when required, by using a greater degree of parallelism in a net organization (Figure 2), the serial-multiplier approach was recommended for development. With a full 32-bit floating point (24-bit mantissa, 8-bit exponent) capability total of 10 of the custom LSI CMOS/SOS serial-type multiplier packages supported by 15 packages accommodating custom LSI control chips, a full six-term multinomial of 2-inputs can be evaluated in 5.2 microseconds. For 16-bit fixed point capability the control chips reduce to two, and the computation period reduces to 2.9 microseconds. The random-access memory to support each element will depend on its degree of multiplexing. For a fully populated array, up to 3 RAM packages would be required per element.

The advantages of such arrays include:

1. High-speed capability for general-purpose digital processing. The speed can be tailored by providing varying degrees of parallelism of the element in an array.
2. Capability for relieving software problems. The array can be considered as firmware once a set of coefficient and interconnect vectors have been found, or are known, for a particular application.
3. Capability for providing fault-tolerant computing. In general, alternate coefficient and interconnect vectors exist to realize the same transformation. Hence, if an element in a fully populated array is faulty, reprogramming of the array is possible by by-passing any faulty element(s). The degree to which this by-passing is done will, of course, depend upon the number of elements available and the complexity of the transformation.

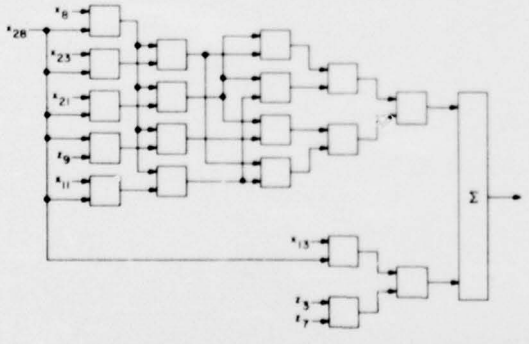


Figure 9—Multinominal network classifier that discriminates between two languages. The x's and z's are phoneme and phoneme-like parameters from the special waveforms

ACKNOWLEDGMENTS

The guidance provided for this program by Mr. C. W. Gwinn of the U.S. Air Force Avionics Lab. is appreciated. Portions of this work have been performed for the Air Force Systems Command, United States Air Force, under Contract F33615-73-C-1089 to RCA with Adaptronics, Inc. subcontractor. The other portions of this work have been performed by Adaptronics, Inc. under the sponsorship of Armco Steel Corporation, RCA, the U.S. Air Force, and other organizations. Assistance of H. Urkowitz and D. A. Miller of RCA and A. N. Mucciardi of Adaptronics, Inc. in aspects of this work is acknowledged.

REFERENCES

1. Gilstrap, L. O., Jr., H. J. Cook and C. W. Amstrong, *Study of Large Neuromime Networks*, Adaptronics, Inc. Fin. Rept. Contract

AF33(615)-5125, AF Avionics Lab., AFSC, AFAL-TR-67-316, AD#824 470, Dec. 1967.
2. Gilstrap, L. O., "Keys to Developing Machines with High-Level Artificial Intelligence," presented to *ASME Design Conference*, New York, April 22, 1971, (ASME paper #71-DE-21).
3. Cleveland, D., "Hardware Realization of Trainable Multivariable Nonlinear Transformations," presented to *Computer Designers Conference*, Anaheim, Calif. January 19-21, 1971.
4. Mucciardi, A. N., "Neuromime Nets as the Basis for the Predictive Component of Robot Brains," presented to *ASC Fourth Annual International Symposium*, Washington, D.C., October 8-9, 1970; *Cybernetics and the Management of Ecological Systems*, Robinson (Ed.), Spartan Books.
5. Ivakhnenko, A. G. et al., "Group Handling of Data in Identification of the Static Characteristics of a Multi-Extremal Plant," *Soviet Automatic Control*, 14, 2, 1969, pp. 30-37.
6. Ivakhnenko, A. G. and Y. V. Koppa, "Stochastic Algorithms and the Group Method of Data Handling in Prediction of Random Events," *Soviet Automatic Control*, 14, 3, 1969, pp. 20-32.
7. Ivakhnenko, A. G. and Y. V. Koppa, "Algorithm of the Group Method of Data Handling Using Linear Operators," *Soviet Automatic Control*, 14, 4, 1969, pp. 50-58.
8. Ivakhnenko, A. G., "Polynomial Theory of Complex Systems," SMC-1, 4, October 1971, pp. 364-378. *IEEE Trans. on Systems, Man and Cybernetics*.
9. Hampel, D., R. W. Blasco and D. Cleveland, "Electronically Programmable LSI Arrays," *NAECON '74 Record*, pp. 134-141.
10. Gilstrap, L. O., Jr., "An Adaptive Research to Smoothing Filtering and Prediction," *Proc. of 1969 NAECON*, pp. 275-280.
11. Barron, R. L. and C. W. Gwinn, "Applications of Self-Organizing and Learning Control to Aeronautical and Industrial Systems," presented to *ASME Design Engineering Conference*, New York, April 19-22, 1971. (ASME paper #71-DE-22).
12. Barron, R. L., "Theory and Application of Cybernetic System: An Overview," *NAECON '74 Record*, pp. 107-118.
13. Hampel, D., L. Micheel, K. Prost, "Threshold Logic Multiplier," *NAECON 1973 Record*, pp. 288-295.
14. McIver, G. W., R. W. Miller, T. G. O'Shaughnessy, "A Monolithic 16×16 Digital Multiplier," *ISSCC Digest*, Feb. 1974, pp. 54, 55.
15. Barron, R. L., "Applications of Learning Networks in Computer Aided Prediction and Control," *SME CAD/CAM Conference*, February 12, 1975.
16. Mucciardi, A. N., R. Shankar, J. Cleveland, W. Lawrie and H. L. Reeves, *Adaptive Nonlinear Signal Processing for Characterization of Ultrasonic NDE Waveforms*, Interim Report, Task 1, Inference of Flat-Bottom Hole Size, Adaptronics, Inc. January, 1975.
17. Barron, R. L., F. W. van Straten, and R. F. Synder, "Inference of Refractivity Structure and Generation of Ray Traces by Analytical and Adaptive Methods," *1975 NAECON*, To be Presented.
18. Mucciardi, A. N., *Demonstration of Capability of Adaptive Learning Networks to Discriminate Automatically Between Earthquakes and Nuclear Explosions Based on Seismic Parameters*, Adaptronics, Inc., ATN-76, April 23,1973.
19. Mucciardi, A. N., E. C. Orr and R. Shankar, *Discrimination between Five Spoken Languages by Trainable Hypercomp™ Network Classifier*, Adaptronics, Inc. Final Report for RCA Corp., April 22, 1974.

APPENDIX F

APPLICATION TRENDS FOR ELECTRONICALLY PROGRAMMABLE ARRAYS

D. Hampel
RCA
Government Communications and Automated Systems Division
Somerville, N.J.

R.L. Barron
Adaptronics, Inc.

ABSTRACT

In the past two years, significant advances have been made in parallel array processing, particularly in the two-dimensional programmable arrays. These arrays are based on elements capable of evaluating, numerically, quadratic multinomials of two variables. Specifically, a degree of advanced LSI technology has been applied to the arithmetic unit of such elements, and some of the control problems were addressed and design approaches developed. A number of new application areas were identified and work was done to demonstrate solutions with the array techniques.

The relationship between these polynomial modeling arrays and other parallel array processing techniques is established. Then, the concepts behind these arrays that deal with their use in adaptive modes as well as in more conventional high-speed signal processing are discussed. Degrees of hardware that can be applied in simulating and implementing these arrays are shown. Concepts of multiplexing, distributed memory and full parallel processing are discussed along with appropriate circuit-technology impact. Finally application areas are given.

INTRODUCTION & BACKGROUND

This paper deals with arrays (physically realizable as one-, two-, or three-dimensional) of computing elements which can be programmed to solve or model particular input-output relationships. Various terms have been associated with such arrays, including:

distributed logic/memory processing[1]
associative processing[1]
cellular arrays[2,3]
programmable arrays[4]

The element (or primitive function) is, in effect, the basic building block of such arrays and can be given any one of a variety of characteristics. Generally, the element contains logic and memory circuits such that the output is a function of stored coefficients as well as the input data set. They can be given the ability to operate on 1) binary variables,[2,3] 2) numerical variables (binary encoded)[4,5,6] or 3) analog variables.[7,8,9,10] Numerical elements restricted to evaluating a given type or set of expressions, such as polynomial expressions, will be the topic of discussion in this paper.

Each of the three types of elements can theoretically be structured in arrays of arbitrary complexity. The advantages of such arrays, compared to single element realizations, include the possibility of higher throughputs, and fault tolerance.

The structure of arrays is demonstrated in Fig. 1, which shows examples of one- and two-dimensional nets. An example of a one-dimensional array useful for signal processing is a transversal filter, Fig. 1(a). In this application each input is multiplied by a weight (the element function is multiplication) and the outputs are summed. The fact that the inputs are frequently time delayed samples of a given waveform imparts the additional requirement on the element of shifting. Such "arrays" can be realized by CCD's or Surface Acoustic Wave (SAW) devices (for analog signal processing), binary correlators (for binary or clipped signals) or arithmetic units as in a digital filter operating on A/D converted signals.

Two examples of two-dimensional array formats are shown in Figs. 1(b) and (c). In the first case, the elements are arranged in layers and information flows from the input data set to the output. Each element's output goes to an input in the next layer. An example of this type of array is the polynomial evaluator net now undergoing design and development with special LSI circuits; this net is the main subject of this discussion on parallel array processing. Another two-dimensional structure is the so-called cellular array proposed for binary image processing, where each element (or cell) communicates with each surrounding cell.[2,3] Data are manipulated according to "surround patterns" and a stored base of algorithms. Information is entered in two-dimensions and processed in two-dimensions.

There is one more important aspect to such arrays, which has in fact been responsible for much of the research effort expended for their implementation and application. This is their potential for being reconfigured to suit a number of different problems, or, when used for a given transformation, to be adapted to changes in the environment by virtue of being able to have their weights or coefficients modified. In describing these processes the array type of Fig. 1(b) will be used as an example, although analogies exist for all types of arrays.

It is assumed that each element's output Y is a function of its inputs (X_I's) and some constants or weights (W_i's), and furthermore, chat each element in a layer can be interconnected to any

consists of N observables, $x_1, x_2, \ldots, x_N$. Also suppose that the output is a scalar whose value may be considered as the estimate of some property of the input process. In general, y will be some nonlinear function of the x_i's as follows:

$$y = f(x_1, x_2, \ldots, x_N) \qquad (1)$$

POLYNOMIAL (MULTINOMIAL) APPROXIMATION

Under fairly general conditions, a function of N variables may be expressed in an N-dimensional Maclaurin series as follows:

$$y = a_o + \sum_{i=1} a_i x_i + \sum_{i=1}\sum_{j=1} a_{ij} x_i x_j + \sum_{i=1}\sum_{j=1}\sum_{k=1} a_{ijk} x_i x_j x_k + \ldots \qquad (2)$$

In the most general case, the coefficients, a_o, a_i, ..., are functions of time, but for many cases of interest, the underlying characteristics of the x's do not depend on time and consequently the coefficients are constants.

A trainable network may be used to implement Eq. 2. A basic element of the network is the two-input, single-output device illustrated in Fig. 2(a) (and previously alluded) to which implements the following function (P3) of its input x_1, x_2:

$$y = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_1 x_2 + w_4 x_1^2 + w_5 x_2^2 \qquad (3)$$

NETWORKS OF THE BASIC ELEMENT

Consider now two layers of such elements as illustrated in Fig. 2(b). The figure shows that each z_i contains pairwise products up to degree four. Note that the first layer contains all possible pairs of three inputs x_1, x_2, x_3. To implement a general multinomial expression (i.e., a polynominal in many variables), the number of elements in

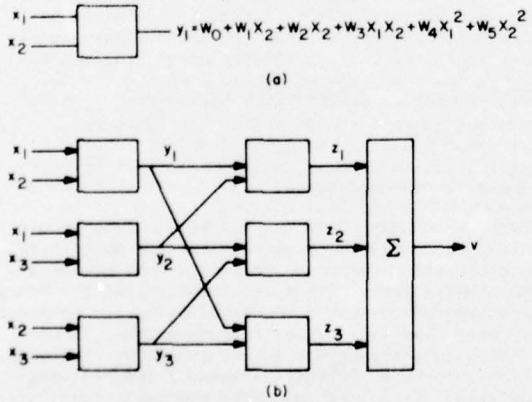


Fig. 2 - (a), Basic element; (b), 2-layered network of elements.

each layer would have to grow as one proceeds deeper into the network. However, it is found empirically that acceptable approximations are obtained without this growth; in fact, the number of elements in successive layers will soon (after, perhaps, two or three layers) decrease, until only a few are left as inputs to the final network component (which is an adder).

THE KNOWN DATA SET

The matter of determining the coefficients of each network element and the number and interconnections of the elements are now considered. These tasks are accomplished with a "known" data base; that is, a data base for which the values of the dependent variable are known. The steps involved are:

1. Optimization of the coefficients in each element of the first layer,
2. Selection of those elements whose output is acceptable (rejection of poor performers),
3. Repetition of the process for each layer, and
4. A global optimization (adaptation) of all coefficients in all layers based upon the final output.

The known data base is divided into three independent but statistically similar subsets:

1. Fitting subset
2. Selection subset
3. Evaluation subset

The fitting subset is used to determine the coefficients of the elements. The selection subset is used to reject the poor performers and to determine when to stop evalution of the network. The fitting and selection subsets are also used for the global optimization. The evaluation subset is used to estimate the overall performance. Since the evaluation subset was not used for network synthesis, the performance on this subset is an accurate estimate of the ability of the network to generalize to new, previously unseen data.

TRAINING THE NETWORK

The element coefficient determinations are based, in part, upon a least-squares fit to a desired output. Other criteria are, of course, possible, and are often used. Employing a least-squares criterion, the elements are first adjusted by a matrix algebraic procedure and then by a recursive search (i.e., optimization) procedure. An outline of the steps follows.

The fitting and selection subsets are used alternately in training each layer. The fitting is used first to establish the coefficients. The specific observables to be used initially have already been chosen. Let these be designated by $x1, \ldots x_N$. These observables, are arranged in pairs, x_i, x_j; where i, j = 1, ... N. There are N(N-1)/2 such pairs. Thus, the first trial will require N(N-1)/2 trainable elements (such as that shown in Fig. 2(a)). A pair of observables is sent to each element. The

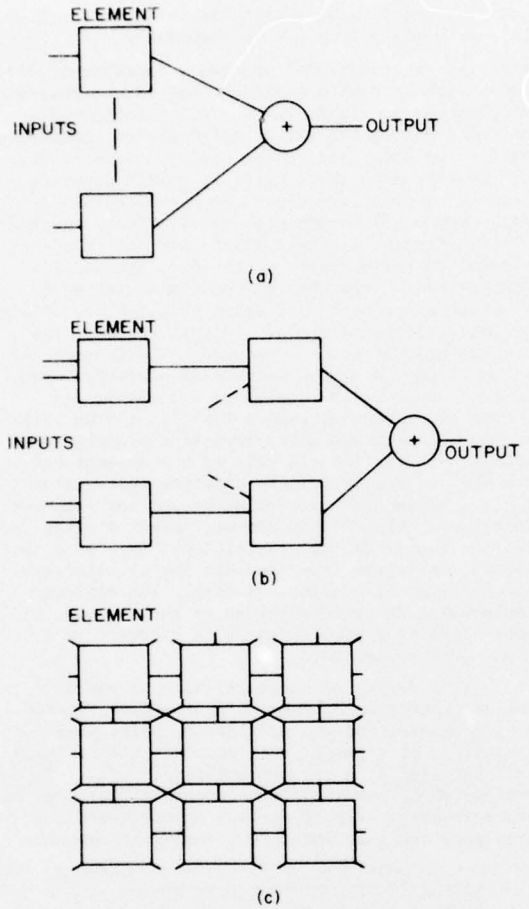


Fig. 1 - Parallel array types: (a), 1-dimensional; (b) and (c), 2-dimensional.

element in a preceding or succeeding layer. Then the overall array can synthesize a wide variety of transformations between the inputs and output(s)*, limited only by the element's primitive function and the total number of elements. The features of weight variability and interconnect flexibility allow such nets to be trained or to fit a desired input-output characteristic, as will be described later. Then, a given numerical processing array, for example, could evaluate a known transformation such as an FFT, as well as heuristic, nonlinear transformations found by search algorithms.

The problems associated with parallel array processing include:

1) Reasonable hardware implementations. To capitalize on the potentials of such arrays for high-speed signal processing, latest-technology LSI must be incorporated. The first generation of parallel array processing approaches was limited in application because of the relatively large numbers of components required to populate an array. Hence, most work on arrays has been done in software simulation of the actual hardware concepts.

2) Control. The problems of data entry, reconfigurability, supervisory control, and system communication must be addressed.

3) Application. Many of the problems which can benefit from parallel array processing are frequently, in essence, re-stated so as to fit more conventional solutions. These problems must be examined in a new light to effectively apply the new techniques.

The methodology for training, trends in hardware realization of such arrays, and applications will be discussed.

TRAINING AND ADAPTION OF MULTINOMIAL NETWORKS

The methods for training and adaptation of algebraic multinomial networks that are constructed from the arithmetic elements described above are now considered.[11,12,17] This is done with special algorithms, generally off-line, that specify network configurations. Applications of multinomial networks generally involve operations with sensor data that are obtained as a result of "observing" a physical object, process, or phenomenon. The classical approach to design of computer models for inferences and predictions from observations has been to determine all the relevant characteristics, deterministic and/or statistical, of the process being observed, and to use these measurements (and assumptions) in design calculations.

Very often the structure of the model is presumed and the design takes the form of calculating the values of certain parameters. Even if the nature of the observed process changes, the structure of the model often does not change, but the parameter values are adjusted in response to measured changes in the inputs or outputs.

In many important applications, the inputs (i.e., the observables) are difficult to describe analytically. The best or even a good structure for the model cannot be determined a priori. In this case, it is desirable to have a model structure that can adjust to representative inputs. That is, the model is trainable, both in it structure and in its parameter values.

It is therefore desired to implement a general (usually nonlinear) function of certain input variables which we can call observables. Since little may be known about the characteristics of the observables, the parameters of the network are not known a priori. The network will have to be trained with representative inputs. To make the ideas clear, suppose that the input

* In general, nets may have more than one output.

coefficients of the element are determined using a least-squares criterion procedure. The procedure is repeated for each of the N(N-1)/2 elements. Not all of the pairwise combinations are of significant aid in extracting the desired information. The selection process, using the selection subset, eliminates those elements whose performance is not acceptable. The performance may be measured by the square of the error magnitude. Now there are, perhaps R elements which survive. The process is repeated for the second layer. This layer starts with (R) (R-1)/2 elements. The training subset of observables, restricted to those which survived the first selection process, is inserted into the first layer, resulting in R(R-1)/2 pairs of inputs into the R(R-1)/2 initial elements of the second layer. Then the selection subset is inserted into the first layer and the selection process is applied to the second layer. This procedure will eliminate the unacceptable pairs from the second layer inputs.

The process is repeated with succeeding layers until the error rate on the selection subset is minimized. Although further reductions in the error rate on the fitting subset are realizable by incorporating more layers, to do so would produce over-fitting of the fitting data. Eventually, a single output results from each of several disjoint sub-networks. These outputs are added to produce a single output. There is a final step in the training process. This is a process of vernier adjustment, or "fine tuning" of the coefficients.

Once a process is modeled, or a function fitted by computer algorithm, the hardware elements (array) described previously can be programmed for high-speed real-time evaluation of new data.

DEGREES OF ARRAY HARDWARE

In the simulation and realization of complex polynomial modeling nets, the degrees of hardware (and associated speed improvement) are:

1. All - software simulation
2. Hardware multiply
3. Polynomial evaluator
4. Element
5. Layer of elements (two or more elements)
6. Full array (two or more layers)
7. Adaptation algorithm hardware

Most of the work, to date, in array processing has been done in software (1). Whether for training or for actual control and classification, software implementations result in inordinately long periods of calculation and hence cost. Software, which is impractical for all but the slowest speed control problems, has spurred the developemnt of hardware to effectively deal with complex modeling and transformation problems at lower costs and/or in real time. Of course other advantages will accrue with the hardware, such as improved reliability due to redundancy.

The first and most basic hardware improvement which can be imparted to a computer used for polynomial modeling using arrays is the incorporation of a hardware multiplier (2) with associated input-output buffer registers and control logic. This feature can improve a software multiply period anywhere from 50 to 500 times depending on the computer (microprocessor or miniprocessor). Next, instead of transferring a single multiplier and multiplicand to the hardware multiplier, groups of numbers can be transferred for evaluation of a complete polynomial expression (3), thus shortening overall transfer periods. In this concept, two or more multipliers can be used. The element (4) is an extension of the polynomial evaluator, and is defined as being capable of evaluating any number of polynomial expressions[19], storing intermediate results and using them in a preprogrammed manner. The arithmetic unit of the element has a number of multipliers to expedite calculations, since a minimum of time is necessary for computer interface. The element has sufficient storage for all the inputs (x_i's), coefficients (w_{ij}'s) outputs (y_i's), and connection controls (c_i's) necessary for an array of a given capacity. The minimum improvement in speed provided by the element is about 1000 to 1 when compared to software on a high-speed processor.

Up to this degree of hardware (4) a simple I/O bus structure can be used to communicate between host processor and the multiplier, polynomial evaluator, or element. For concepts (5) or (6), full parallel array processing is possible. To achieve this, two techniques are required: loading and switching. Fig. 3 shows a multi-layered structure being loaded from a common bus by the

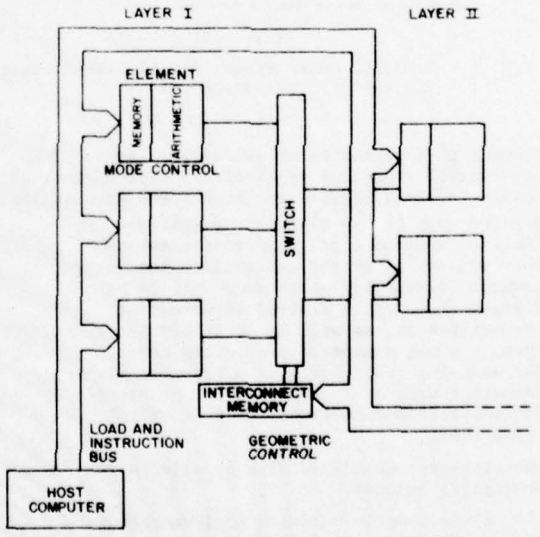


Fig. 3 - Loading and controlling a parallel array.

host computer. Layer 1 outputs are switched to appropriate layer 2 inputs under the space-division

switch memory control (also pre-loaded by the host computer).

The speed improvements inherent with multi-element arrays compared to a single multiplexed element are proportional to the number of elements populating an array and depend on whether multi-layer implementations can be used in a pipeline mode to further increase throughput. With maximum employment of LSI for the element realization, approaches (5) and (6) become viable alternatives for improved-performance, parallel array processing. Since each element in the numerical polynomial processing concept can handle two inputs, the number of elements in a layer, for maximum speed, for single output nets, depends on the total number of observables or variables bearing on the problem, divided by two. Most previous applications have indicated a maximum requirement of 20 elements in the first layer. In the distributed element case, (5 and 6), the total memory requirements are the same as for the multiplexed or time shared case, except that the memory is physically partitioned along with the arithmetic function. This arrangement will lead to different memory organizations.

For example, whereas a moderate size RAM will be used with a single time-shared element, smaller register stacks would be provided with a distributed element structure, as part of their arithmetic units. This would further simplify buffering and interconnections.

ADAPTATION ALGORITHM HARDWARE

An array can be considered a high-speed numerical calculator for deterministic type problems (problems whose input-output relationships or transformations are known a priori). An FFT is an example of a known, useful linear transformation. More interestingly, and of greater importance, the array concept can be used to synthesize arbitrarily complex linear or nonlinear expressions and to help find (in a training or adaptation mode) a suitable relationship between a set of input data and a known output result, as described previously. Once a relationship is found, the array can be used in its deterministic mode appropriately responding to new sets of data from the same physical system in providing desired outputs. In the adaptation mode, the hardware element(s) works very closely with the host computer, which has been pre-loaded with algorithms for random search of the discriminating hypersurface, for example. Here the elements are used as an aid in speeding up the adaptation approach which is, basically, a trial and error procedure requiring a large number of calculations to "home in" on a desired or acceptable relationship.

Various amounts of hardware can be applied to the adaptation mode to alleviate the computer-element communications problem, where, after each trial, the host computer must determine a new set of weights for use in the next trial, etc. For these types of algorithms, random-number generators can be provided, associated with the element, with appropriate control logic and memory, so that in the array adaptation mode, input data are paired and element weights adjusted automatically and rapidly, in an effort to minimize errors between known, desired responses and the calculated output responses. Weights derived from random-number generators can be incremented or decremented by given amounts according to preset or manipulated statistical distributions. This type of hardware can eventually be coupled to the array itself to augment high-speed self-adjustable transformations, replacing the software algorithms.

HARDWARE FACTORS

The overall hardware requirements of a programmable array capable of complex polynomial modeling will depend on throughput requirements. The other factors influencing hardware complexity are:

1. Precision of numerical calculations
2. Fixed- or floating-point notation
3. Total memory requirements
4. Level of LSI applied
5. Calculation speed

Shallow nets with relatively few layer nets do not require the precision of deep, highly nonlinear nets, and can use 12- to 16-bit, fixed-point numbers. However, for complex situations, after the third or fourth layer, 32-bit, floating-point representation becomes important. Hence, an element with varying capability would be desired to optimize its use within an array. Since loading an array's memory from the host processor can consume large periods of time, the array's memory, whether locally concentrated for single elements or distributed for a fully populated array of elements, should have the capacity for as many problems as the unit will be called upon to solve in a given system. For example, the same array might serve as a classifier for two or more problems, where the coefficients and element interconnectivity are all different. If the array's memory had all the necessary values stored for both problems, a single command could have the array provide classification for either problem with virtually no added delay as a result of problem changeover.

TECHNOLOGY CONSIDERATIONS

What technology should polynomial arrays use for optimum performance? LSI permits us to consider fully populating large parallel arrays for high throughputs. However, the partitioning of the array element into its constituent functional parts must be based upon a realistic assessment of both available and developmental technology trends. This is particularly true for the memory and arithmetic portions of the element. The fundamental processes of multiplication and addition must be efficiently implemented and matched to supporting memory for accessing data and coefficients (or weights).[4] It was shown[19] that the profusion

The network for flat-bottom hole defect inference consists of 13 elements containing a total of 78 coefficients. The network implements an eight degree function of the 15 input variables.

Work is now proceding on inference of the size of fatigue-induced cracks emanating from fastener holes in specimens representative of wing structures. The problem in the use of ultrasonic inspection is that small defects, such as fatigue microcracks, may be masked by reflecting surfaces of the fastener and the part itself. Adaptively trained networks offer an attractive approach because they do not rely on theoretical formulations for these complex signal environments.

MATERIAL TECHNOLOGY

Potential applications of adaptively trained nonlinear networks in materials technology include:

- Characterization of properties of composities and of adhesively-bonded assemblies
- Inference of material physical properties from microstructure data
- Productivity modeling
- Monitoring and optimization of processes for material removal
- Predictive control of melting, forging, casting, extruding, and rolling processes
- Forecasting of maintenance necessitated by corrosion of flight vehicle structures
- Inference of NDE effectiveness of facilites by their performance on test sample

In each of these instances, the key point is that networks can learn to infer or predict from the natural data that are produced by real processes. These "natural" data may be records of sounds, vibrations, etc. -- anything readily accessible for economical instrumentation or recordkeeping.

PHYSIOLOGICAL MONITORING

Adaptive nonlinear network modeling techniques have been successfully applied to automatic interpretation of animal and human EEG's and other physiological waveforms. It is conceivable that future manned systems will use trainable multinomial networks to alert crew members and/or remote medical personnel to subtle changes in alertness and psychological and physiological readiness of the crew.

DIGITAL SIGNAL PROCESSING

As noted elsewhere in this paper, adaptive programmable networks can potentially implement an extremely wide variety of linear and nonlinear signal processing functions. These functions include:

- Auto-and cross-correlation functions
- Fast Fourier transforms
- Power spectra and cepstra
- Digital bandpass filtering
- Extraction of time derivatives
- Transversal filtering
- Inference of waveform nonlinear parameters
- Nonlinear predictive coding

By combining these (pre-processing) network functions with other network functions that implement the main classification, prediction, decision, and/or control logic, the functional possibilities became boundless. Clearly, the trend will be toward realization of very fast LSI arrays so that the array hardware can be multiplexed rapidly to realize powerful processing combinations.

TRAJECTORY PREDICTIONS

Successful R&D has been conducted for more than a decade by application of multinomial networks to prediction of aerospace vehicle trajectories. In summary, these investigations have established that the network methods are capable of inferring vehicle parameters, such as ballistic coefficients, quite accurately, and that the networks can make extremely fast, accurate predictions of ballistic trajecteries. These predictions are comparable in accuracy to the conventional procedures where equations of motion are integrated in erial computers, but are very much faster because of the computing speed of the parallel network structure.

TARGET SIGNATURE CLASSIFICATION

Seismic and/or acoustic waveforms from remote sensors on the ground can be used with nonlinear networks to detect and classify accurately the presence of different types of ground vehicles, aircraft, and of personnel.

RADAR REFRACTIVE-INDEX CORRECTIONS

A nonlinear adaptive network can be used to fit radar metric data on the range, elevation, azimuth, and true height of a cooperative target or targets. After fitting a network model to these data, the model may be interrogated to estimate true heights of any other targets observed within the bounds of the range, elevation, and azimuth training of the network. By taking care not to overfit the model during its training, it will be a smoothly-fitted functional approximation. From the mathematical standpoint the model will be continuous and differentiable, and its derivatives will be close approximations to the quantitative derivative behavior of the physical process (i.e., ray bending as a result of refractive-index gradients within the atmosphere). Numerical partial derivatives of the form $\partial y/\partial x$ can be used to determine the slope of the ray as it traverses the anomalous part of the atmosphere, presumably the part that has been adaptively modeled. Knowing this slope, the path of the ray beyond the modeled region may be estimated by conventional procedures.

The results in reference 14 show that a substantial improvement in accuracy of target height determinations can be achieved using the network modeling method in comparison with the usual method for refractive-index corrections.

DETECTION OF REMOTE NUCLEAR EVENTS

Reference 15 presents results of work performed

of multiplication functions required for complex polynomial modeling could best be implemented with a number of LSI serial/parallel type multipliers in a bit serial word parallel fashion rather than parallel array type multipliers which, because of their complexity, would have to be used in a bit parallel - word serial fashion. This scheme provided a minimum calculation time complexity factor (an architectural figure of merit), while ultimately being able to approach speeds of all-parallel multiplier implementations.

The serial nature of the basic multiplier simplifies scaling for floating point (exponent control) and addition, as well as minimizing pin connections. Furthermore, because of the predominantly "on-chip" nature of the multiplication logic (a single output driver only being required), the CMOS/SOS technology offered an ideal medium for the LSI implementation. Its relatively high packing density and low power dissipation resulted in a full 24-bit multiplier-accumulator (capable of an (ax + b) calculation) on only a 150- by 170-mil chip, leaving room for substantial additional logic or memory growth, while still staying within the realm of high yield technology. Alternate technologies could improve speed at a high chip count (cost) and power dissipation. An ECL implementation could improve speed by factors of three to five, but chip power and area constraints would inhibit the size of the multiplier to about eight or twelve bits (instead of 24) and preclude the ability to add associated circuitry.

Another consideration in the element architecture is the required memory and its interrelationship with the multipliers in executing the polynomial calculations. Each element in an array (physical or simulated) must have a store containing all the coefficients, data and commands it will need for a given transformation or set of transformations for rapid recall at the appropriate time. This store is necessary to minimize "outside World" interaction, since the relatively slow rates at which a host processor can load new sets of values into the array would defeat the purpose of the high-speed processing capability of the array.

Presently, array designs are being based on the use of MOS RAM's, providing access times in the order of 100 nanoseconds. Future trends will lead to the use of CCD serial memories, whose characteristics will optimally fit the array architecture, at projected cost for memory of about 1/10 the cost of advanced RAM's.[13] An array organization based on the use of CCD memory is shown in Fig. 4. Each element's memory would be loaded with all the w's it will need to solve the problem(s) that the array will be used for. These w's must of course be preloaded in the proper sequence by the host computer. Interface logic and buffering requirements are minimized because the serial nature of the multipliers and memories are compatible. With one or two chip types used for multiplication and associated control, and with available, low cost, memory circuits, a wide performance spectrum of parallel arrays can be implemented.

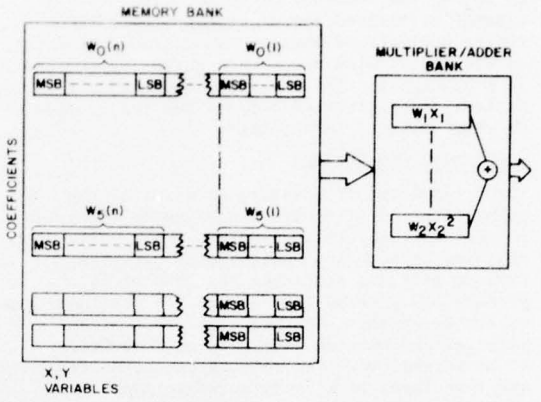


Fig. 4 - Serial memory-bank organization for programmable array.

APPLICATIONS

Aerospace applications of the array/network concepts presented above include:

- Nondestructive evaluation of structural parts[18,21]
- Materials technology[20,22]
- Physiological monitoring
- ECM and ESM systems
- Digital signal processing
- Trajectory predictions[23]
- Target signature classifications
- Radar refractive-index corrections[14]
- Detection of remote nuclear events[15]
- Voice data processing[16]
- Reconnaissance image processing
- Avionics information systems

A number of these areas are briefly described below.

NONDESTRUCTIVE EVALUATION

It has been shown that an adaptively trained, nonlinear, multinomial network provides accurate inferences of flat-bottom holes sizes in ultrasonic nondestructive evaluation of test specimens. The waveforms analyzed were ultrasonic pulse echoes obtained from two different sets of 7075-T6 aluminum area-amplitude test blocks and three different transducers. The eight flat-bottom hole defect sizes ranged from 1/64- to 8/64-inch in steps of 1/64-inch.

The 15 input parameters found to be most informative were those that describe the overall shape and content (area) of parts of two waveforms computed from each pulse echo waveform: the power spectrum and its log Fourier transform, the cepstrum. Maximum amplitude of the echo was found not to be a discriminating parameter when the material and/or transducer were changed.

to assess the accuracy of an adaptive network classifier trained for discrimination between remote underground nuclear events and deep-core earthquakes (which masquerade as nuclear events in many cases). The results show that nearly perfect discrimination between the two classes of remote events is obtained.

VOICE DATA PROCESSING

The application of adaptive networks to the identification of spoken languages is discussed in reference 16, which presents the results of an investigation into multinomial networks used to generate nonlinear features of 29 phoneme and phoneme-like parameters obtained from speech waveforms. The networks are trained to discriminate between each pair of languages to be identified. The outputs of the networks are then input to a decision logic that identifies which language is being spoken. In the case of five languages to be discriminated, this means that a group of nonlinear transformations is produced by ten individual networks, each of which maps the 29-dimensional input space into its respective compont of a ten-dimensional output space. The structure of a typical trained network is shown in Fig. 9. The results of the cited investigation show that significant improvement is obtained in the accuracy of language identification and in insensitivity to the idiosyncracies of individual speakers.

AVIONICS INFORMATION SYSTEMS

The authors believe the trend of adaptive network applications is in the direction of a versatile central avionics processor built around a programmable network that performs many portions of the avionics processing task. The central processor will be fed by remote units that perform signal conditioning and pre-processing (parameterization) of signals at their sources. These remote units may use small programmable networks. The central processor will have, perhaps, 200 to 300 hardware algebraic elements in a network that may be switched almost instantly (via distributed storage of pre-learned connectivity and coefficient information) between a large number of uses. These uses might include fault monitoring; resources management; real-time interpretation of multisensor data; navigation, guidance, and control; physiological monitoring; EW functions; and signal analysis.

SUMMARY AND CONCLUSIONS

Methods for implementing and applying electronically programmable arrays for numerical processing have been explored. These arrays are a subset of parallel array processors in general, which include cellular arrays operating on binary signals. The distinguishing feature of the electronically programmable arrays explored in this paper is their ability to perform complex modeling (linear or nonlinear) of an input data set to achieve a given output result. The rationale and ways of synthesizing and using such trainable multinomial networks were described. The network realization choices were put in perspective, from software simulation to all hardware array population for maximum throughput. The impact of advanced technology in memory and logic was then related to the programmable array hardware. Finally, a number of application areas were described, ranging from inspection of structural parts to high speed electronic warfare signal processing.

It is felt that this type of programmable array, now undergoing development with advanced LSI, will find increasingly greater application. It's inherent ability to be structured into a variety of configurations to efficiently meet a broad range of speed requirements (by trading off hardware complexity) and its ability to perform known linear as well as nonlinear transformations (derived by training) imparts a universality to the programmable array.

The transformations which these arrays realize can be used for application in the areas of signal classification, pattern recognition, or system control.

ACKNOWLEDGMENTS

The guidance provided for this program by Mr. C.W. Gwinn of the U.S. Air Force Avionics Lab. is appreciated. Portions of this work have been performed for the Air Force Systems Command, United States Air Force, under Contract F33615-73-C-1089 to RCA with Adaptronics, Inc., subcontractor. The other portions of this work have been performed by Adaptronics, Inc. under the sponsorship of Armco Steel Corporation, RCA, the U.S. Air Force, and other organizations. Assistance of H. Urkowitz and D.A. Miller of RCA and D. Cleveland and A.N. Mucciardi of Adaptronics, Inc. in aspects of this work is acknowledged.

REFERENCES

1. Joseph, E.C., "Innovations in heterogeneous and homegeneous distributed-function architectures," Computer, Vol. 7, No. 3, Mar. 1973, pp 17-24.
2. Golay, M.J.E., Hexagonal Parallel Pattern Transformations, IEEE Transactions on Computer Vol. C-18 No. 8 Aug. 1969 pp 733-740.
3. Kautz, W.H., "Cellular Logic-in-Memory Arrays," IEEE Transactions on Computers, Vol. C-18, No. 8, Aug. 1969, pp 719-727.
4. Hampel, D., R.W. Blasco and D. Cleveland, "Electronically Programmable LSI Arrays," NAECON '74 Record, pp. 134-141.
5. Adaptronics Inc. Third Quarterly Progress Report for Armco Steel Corp. (Second Year); Jan. 1, 1972.
6. Cleveland, D., "Hardware Realization of Trainable Multivariable Nonlinear Transformations", presented to Computer Designer Conference, Anaheim, Calif. Jan. 19-21, 1971.
7. Widrow, B., Gupta, N.K. and Maitra, S., "Punishment/Reward: Learning with A Critic in Adaptive Threshold Systems," IEEE Trans. on Systems, Man and Cybernetics, Vo. SMC-3, No. 5 Sept. 1973, pp 445-465.

8. Hoff, M.E. Jr., Learning Phenomena in Networks of Adaptive Switching Circuits, Tech. Rep. No. 1554-1 July 1962 Stanford Electronics Lab.

9. Jones, E.R. "A Progressive Learning Classifier" Tech. Rep. No. 6783-1, Dec. 1966, Stanford Electronics Lab.

10. Amari, Shun-Ichi, "Learning Patterns and Patterns Sequences by Self-Organizing Nets of Threshold Elements," IEEE Transactions on Computer, Vol. C-21, No. 11, Nov. 1972, pp 1197-1206.

11. Barron, R.L., "Theory and Application of Cybernetic System: An Overview," NAECON '74 Record, pp. 107-118.

12. Barron, R.L., "Applications of Learning Networks in Computer Aided Prediction and Control", SME CAD/CAM Conference, February 12, 1975.

13. Martin, R.R., Frankel, H.D., "Electronic Disks in the 1980's," Computer, Feb. 1975.

14. Barron, R.L., van Straten, F.W., Snyder, R.F., "Inference of Refractivity Structure and Generation of Ray Traces by Analytical and Adaptive Methods," 1975 NAECON

15. Mucciardi, A.N., "Demonstration of Capability of Adaptive Learning Networks to Discriminate Automatically Between Earthquakes and Nuclear Explosions Based on Seismic Parameters," Adaptronics, Inc., ATN-76, April 23, 1973.

16. Mucciardi, A.N., E.C. Orr and R. Shankar, Discrimination between Five Spoken Languages by Trainable Hypercomp™ Network Classifier, Adaptronics, Inc. Final Report for RCA Corp., April 22, 1974.

17. Hampel D., R.L. Barron, D. Cleveland, "Design and Application of Electronically Programmable LSI Arrays," NCC 1975, AIAA Session.

18. Mucciardi, A.N., R. Shankar, M.J. Buckley "Applications of Adaptive Learning Networks to Nondestructive Evaluation Technology," 1975 NAECON

19. Hampel, D., R.W. Blasco, R.L. Barron, "LSI Electronically Programmable Arrays,"Interim Report 1089-1, RCA and Adaptronics, Inc., April 1974.

20. Mucciardi, A.N., "Application of Adaptive Modeling Techniques to Evaluation of Psychotropic Drugs," NAECON, 1974.

21. Muccardi, A.N., R. Shanker, J. Cleveland, W. Lawrie, H.L. Reeves, "Adaptive Nonlinear Signal Processing for Characterization of Ultrasonic NDE Waveforms", Interim Report, Task 1, Inference of Flat-Bottom Hole Size, Adaptronics, Inc. January, 1975.

22. Mucciardi, A.N., E.R. Johnson, and R. Shankar, Development of Nonlinear Cat Sleep Stage Classifier, Adaptronics, Inc. Final Technical Report 689F for Abbott Labs., March 14, 1975.

23. Gilstrap, L.O., Jr., "An Adaptive Approach to Smoothing, Filtering and Prediction," Proc. of 1969 NAECON, pp. 275-280

APPENDIX G

SIX-CLASS SEISMIC/ACOUSTIC SIGNAL CLASSIFICATION USING ADAPTIVE POLYNOMIAL NETWORKS

Anthony N. Mucciardi
Michael F. Whalen
John D. Sanders
Roger L. Barron

Adaptronics, Inc.
McLean, Virginia 22101

Abstract

The performance evaluation of a six-class, single target classifier implemented using adaptive polynomial networks is presented. The classifier was designed to discriminate between six target classes: tracked vehicles, wheeled vehicles, fixed wing aircraft, rotary wing aircraft, personnel, and nuisances. This discrimination is based on signal waveform features computed from co-located seismic and acoustic transducers. The instrument package is remote from the target and signals are detected on each channel passively, as the target moves within a range of signal detectability. The data base used to synthesize and to test the classifier consisted of field-recorded signals. A principal result is that the feasibility of designing a field usable seismic/acoustic signal classifier has been demonstrated. It is shown that an average overall six-class accuracy of 85 percent with good range capability and reasonable site and speed independence is achievable.

Introduction

A major Army requirement is to be able to classify accurately remote targets via passive means. The classifier system is part of the Army's REMBASS (Remotely Monitored Battlefield Sensor System) project. Usually acoustic (microphone) and seismic (geophone) devices are co-located and a classifier must discriminate up to six target classes based on parameters of these two waveforms. The classifier must be reasonably insensitive to signal variations caused by range, speed, and location (i.e., environmental) factors. Once a classification is made, the decision is radioed to a remote station for tactical evaluation.

A program was initiated to demonstrate the applicability of adaptive, nonlinear signal processing techniques for accurate classification of acoustic and seismic waveforms among six target classes.[8,9] Adaptive Polynomial Networks (APNs) were used in a two-way, or pairwise, classification mode and the overall six-way classification was rendered by a voting procedure. The remainder of this paper describes the data base employed, the extracted waveform parameters, the classifier structure and synthesis, and a discussion of the results.

Seismic/Acoustic Field-Record Data Base

The classes for target discrimination are the six shown in Table 1. The class number will be used as a convention throughout the remainder of this paper, i.e., Class 1 denotes tracked vehicles, Class 2 denotes wheeled vehicles, etc.

Table 1: Six Target Classes

Class Number	Type	Abbreviation
1	Tracked Vehicle	TV
2	Wheeled Vehicle	WV
3	Fixed Wing Aircraft	FWA
4	Rotary Wing Aircraft	RWA
5	Personnel	PER
6	Nuisance	NUS

The data base consisted of 671 seismic and acoustic signatures representing six major target categories. Each signature consisted of a data epoch of 10 seconds duration (that is, 10 simultaneous seconds for the seismic and acoustic waveforms). The sampling rate was 2,000 Hz for both waveforms.

Data were recorded at three sites: Yuma, AR, Grayling, MI, and Ft. Bragg, NC, and signatures from Classes 1, 2, and 6 were available at all three locations. Aircraft data, Classes 3 and 4, were available only at Ft. Bragg, and Class 5 data was available only at Yuma. Historically, the most difficult classes to discriminate have been Classes 1 and 2; the majority of records in the data base were from these two classes; 338 of the 671 signatures (about 50 percent) were Class 1, 28 percent were Class 2 and the remainder of the data were spread among the other four classes. (Seventy-two percent of the Class 1 data were recorded at Yuma.)

The signatures generated by different targets are a function of many variables. Of primary importance are target speed and distance from sensor (i.e., range). Many combinations of these two conditions were available in the data base. The speed versus range distribution for land targets varied from 6 to 31 mph and from 0 to 900 for meters for Classes 1 and 2; and up to 100 meters for Class 6 (walking). Similar data for the air targets were 120 to 450 knots for Class 3 and 60 to 100 knots for Class 4. Altitudes of 200, 400 and 600 feet were recorded for both classes.

The geophone employed had a resonant frequency of 7 Hz, a 70 percent damping ratio, and a pass band of about 7 to 250 Hz. The microphone had a flat response from 39 to 40,000 Hz. The acoustic response was filtered to provide a useable pass band of about 20 - 500 Hz. Both seismic and acoustic signatures were digitized at a rate of 2,000 Hz.

One of the main sources of confusion in the seismic channel is due to the filtering effect of the propogation medium -- the earth -- between target and geophone. This varies from site-to-site, so that a given target moving at a fixed speed at some distance from the sensor will produce dramatically different time domain signatures depending on the recording site. The acoustic channel is not affected by environment. Its main susceptibility is due to wind influences. Therefore, each signal channel has somewhat complementary advantages and disadvantages and this is one of the reasons for interest in two-channel classifiers.

The 671 signals were divided into Design and Evaluation subsets as shown in Table 2. The Design set was used to synthesize the classifier. Its performance was tested on the remaining (and unused) 446 records. The records in the Design set were selected at random, and represent about one-third of the data from each class.

Table 2: Compostion of Design and Evaluation Data Subsets

Class	Design	Evaluation
1	113	225
2	64	127
3	18	36
4	12	24
5	12	22
6	6	12
TOTAL	225	446

Waveform Parameterization

The 29 waveform parameters (i.e., features) that were extracted from the seismic and acoustic channels for one 10-second portion of a target run (usually lasting up to 100 seconds) are listed in Table 3. Parameters from each channel appear in two groups because these features were originally defined by the Honeywell (1-18) and Sylvania (19-28) companies.[2,4,5,6] Thus, each 10-second run was represented as a 29-component feature vector.

Table 3: Seismic and Acoustic Waveform Features

	Feature Definition	Counts/ Epoch
Seismic	1. Zero Crossing 1	0-1,240
	2. Zero Crossing 2	0-220
	3. Zero Crossing 3	0-49
	4. Zero Crossing 4	0-105
	5. Time Between Events 1	0-45
	6. Time Between Events 2	0-40
	7. Time Between Events 3	0-14
	8. Time Between Events 4	0-21
	9. Smoothness	0-46
	10. Duty Cycle Consistency	0-397
	11. High Frequency Energy	0-30
	12. Low Frequency Energy	0-40
Acoustic	13. Zero Crossing 1	0-2,518
	14. Zero Crossing 2	0-944
	15. Zero Crossing 3	0-516
	16. Zero Crossing 4	0-328
	17. Duty Cycle Consistency	0-466
	18. Roughness Count	0-467
		Volts
Seismic	19. Low Frequency Energy	0-6
	20. Low Band Envelope Variance	0-6
	21. Wide Band Envelope	0-6
	22. Wide Band Envelope Variance	0-6
	23. Frequency	0-6
	24. Frequency Variance	0-6
	25. Variance of Frequency Variance	0-6
Acoustic	26. High Frequency Energy	0-6
	27. Wide Band Envelope	0-6
	28. Low Band Envelope	0-6
	29. Logarithm of (Acoustic/Seismic) Energy Ratio	Dimensionless

1/ Note that certain of these features are conceptually the same; however, their methods of extraction were different and therefore all the features were included for completeness. The similar features are:

Seismic Zero Crossings (1-4)	- Seismic Frequency (19)
Seismic Smoothness (9)	- Seismic Wide Band Frequency (23)
Seismic Duty Cycle Consistency (17)	- Seismic Frequency Variance (24)
Seismic High and Low Frequency Energy (11 and 12)	- Seismic Wide Band Envelope (27)

It was found from the correlation matrix of the first 28 features that some of the features were highly correlated. An eigenvector analysis showed that only nine uncorrelated variables accounted for over 90 percent of the data variance. So, ten uncorrelated features were defined from the 29 for classifier synthesis:

$$x'_1 = \sum_{i=1}^{28} u_{1i} x_i$$

$$\vdots$$

$$x'_9 = \sum_{i=1}^{28} u_{9i} x_i$$

$$x'_{10} = x_{29}$$

where the x_i (i=1,...,29) are the 29 features of Table 3, u_{ki} is the i^{th} component of the k^{th} eigenvector (i=1,...,28; k=1,...,9). The ten x' variables were used as inputs to the APN classifier.

Classifier Structure

The structure of the classifier is shown in Figure 1. It consisted of two parts: (1) 15 subclassifiers to discriminate between all possible nonrepetitive pairs of classes, and (2) decision logic for rendering one, overall six-way target classification. Each of the 15 subclassifiers is an APN and the methodology assicated with their synthesis has been summarized previously.[1,3] An APN was trained via the methods summarized in these references, using the data described above for each of the fifteen possible target class pairwise decisions.

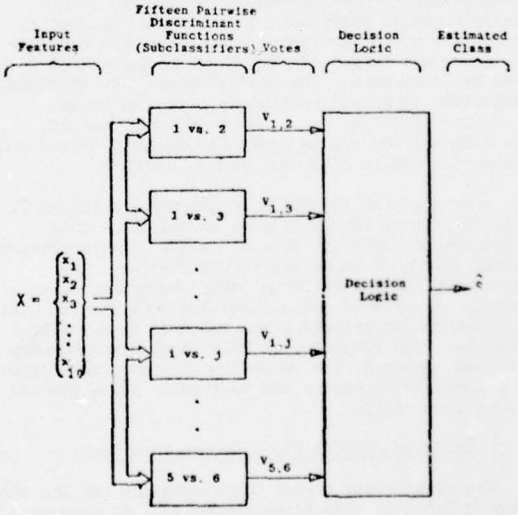


Figure 1: "One-Versus-One" (Pairwise) Classifier Architecture.

Nonlinear pairwise ("one versus one") APN discriminant functions were synthesized by combining pairs of inputs, x_i and x_j, into building-block polynomial elements according to the equation:

$$y = w_o + w_1 x_i + w_2 w_j + w_3 x_i x_j + w_4 x_j^2 + w_5 x_j^2$$

A nonlinear discriminant function may consist of layers of such elements combined to model a given dependent variable. Each layer may consist of as many elements as the number of pairwise combinations of the input

parameters processed by that layer, but only the most discriminating elements need be retained. These elements may then, in turn, be used as inputs to the next layer of the network. The structures of the 15 nonlinear ANN discriminant functions which were synthesized along with the weight coefficients are given in a recent report.[3]

The decision hypersurface of each pairwise subclassifier is such that one target class tends to be mapped into a fixed value below its threshold and the other target class into a fixed value above. The usual convention is that a pairwise discriminant function attempts to map all class i members onto the number +1.0 and all class j members onto the number -1.0, with the discrimination threshold set midway between these two values, i.e., set equal to zero.

In a pairwise test, the proximity of the computed discriminant function output to one of the numbers +1 and -1 (using a suitable metric such as squared normalized difference), the greater the confidence that can be placed in the consequent decision. A tie-breaking strategy that exploits this confidence information is illustrated by means of the following example.

Table 4 contains the hypothetical outputs of the 15 pairwise tests for one 10-second record. It can be seen that with a threshold equal to zero, a positive output renders a unit "vote" for Class i and vice versa for a negative output. (The value "i" is always less than "j"). In this illustrative example, Classes 1 and 2 are tied with four votes each. Classes 3, 4, 5, and 6 are eliminated from further consideration.

Table 4: Illustration of Tie-Breaking Strategy

ANN No.	Class i Versus Class j	Output	Winning Class
1	1 vs. 2	+1.01	1
2	1 vs. 3	+0.91	1
3	1 vs. 4	-0.13	4
4	1 vs. 5	+0.85	1
5	1 vs. 6	+1.15	1
6	2 vs. 3	+0.69	2
7	2 vs. 4	+0.71	2
8	2 vs. 5	+0.87	2
9	2 vs. 6	+0.85	2
10	3 vs. 4	-0.88	4
11	3 vs. 5	+0.92	3
12	3 vs. 6	+1.00	3
13	4 vs. 5	-0.78	5
14	4 vs. 6	+0.91	4
15	5 vs. 6	+0.93	5

	Voting Logic
Target Class	No. Votes
1	4
2	4
3	2
4	3
5	2
6	0

The following measure of confidence was used in the case of ties:

$$M_k = \sum_{m=1}^{K(K-1)/2} V_m \left| 1 - | k_A | \right|$$

where:

K = Number of classes

k_A = Actual output for Class k

V_m = "Selection operation"; $V_m = 0$ if Class k is not involved in the m^{th} test or if k is involved but loses; $V_m = 1$ if Class k is involved in the m^{th} test and wins.

M_k is equal to zero if the actual outputs, k_A, always equal unity. Larger values of M_k usually denote weaker decisions. Therefore, a tie-breaking strategy is to evaluate M_k for those classes k in contention and to choose that class for which the M_k value is smallest.

Continuing, for the example given in Table 4, the associated M_1 and M_2 values for the tied Classes 1 and 2 are:

$$M_1 = |1-1.01| + |1-0.91| + |1-0.85| + |1-1.15| = 0.40$$

$$M_2 = |1-0.69| + |1-0.71| \quad |1-0.87| \quad |1-0.85| = 0.88$$

Therefore, Class 1 is the classified target due to its obtaining the larger degree of confidence.

As an alternative to tie-breaking, it may be desirable to report all classes receiving at least V votes. For example, a classifier can be regarded as having produced a correct response whenever it generates at least V votes.

These and other voting logic procedures have been reported in detail previously.[7]

Performance Evaluation

Two sets of confusion matrices have been generated for the nonlinear "one-versus-one" classifier. The first set, shown in Table 5, was obtained by application of the tie-breaking decision logic; and the second set, shown in Table 6, was obtained by the vote-reporting technique with V = 4, as described above. It was also found that this classifier is reasonably independent of range between target and sensor. Although only a few signatures were available at large ranges, no misclassifications were made for tracked vehicles beyond 600 meters. The accuracies shown in Tables 5 and 6 represent a significant increase over previous efforts.[2,4,5,6]

In evaluating the performance of a classifier, three criteria should be taken into consideration. These criteria reflect the ability of a classifier to perform both accurately and consistently. Thus the performance of the classifier was defined to be a function of three quantities:

1. A - Overall accuracy
2. C - Consistency of overall accuracy
3. S - Site independence

The overall accuracy, A, was defined as the number of correct decisions divided by the total number of decisions for the given classifier. The value of A can range from 0, for total error, to 1, for perfect classification.

Since it is desirable for a classifier to perform equally well for all six target classes, a measure of accuracy consistency, C, was constructed as follows. The average accuracy and its standard deviation, σ, were computed over all six classes, considering all three sites. If the average accuracy was the same for all six classes, σ was zero. Conversely, a large value of σ denoted inconsistent classifications. Therefore, the consistency measure was computed as: $C = 1 - \sigma$. The value of C can range from 0.45, for inconsistent classifications, to 1, for perfectly consistent classifications.

Table 5: Confusion Matrices (Evaluation Data Set) Using Tie-Breaking Logic

Location	True Class	Decision 1	2	3	4	5	6	Class Total	Class Accuracy	Site Accuracy
Ft. Bragg	1	19	8	2	0	0	0	29	.66	
	2	0	21	4	0	1	0	26	.81	
	3	3	2	29	0	0	0	36	.81	
	4	0	2	0	22	0	0	24	.92	
	5	0	0	0	0	0	0	0	-	
	6	2	0	0	0	0	0	2	.00	
										91/117 = .78
Grayling	1	29	8	2	0	0	1	40	.73	
	2	2	40	1	0	0	0	43	.93	
	3	0	0	0	0	0	0	0	-	
	4	0	0	0	0	0	0	0	-	
	5	0	0	0	0	0	0	0	-	
	6	0	2	1	0	0	4	7	.57	
										73/90 = .81
Yuma	1	148	5	2	1	0	0	156	.95	
	2	2	43	4	0	7	0	56	.78	
	3	0	0	0	0	0	0	0	-	
	4	0	0	0	0	0	0	0	-	
	5	0	3	0	0	19	0	22	.86	
	6	2	0	0	0	0	1	3	.33	
										213/239 = .89

Overall Accuracy: 377/446 = .85

Table 6: Confusion Matrices (Evaluation Data Set) Using Vote Reporting Procedures

Location	True Class	Decision 1	2	3	4	5	6	Class Total	Class Accuracy	Site Accuracy
Ft. Bragg	1	20	7	2	0	0	0	29	.69	
	2	0	21	4	0	1	0	26	.81	
	3	3	1	30	0	0	2	36	.83	
	4	0	1	0	23	0	0	24	.96	
	5	0	0	0	0	0	0	0	-	
	6	2	0	0	0	0	0	2	.00	
										96/117 = .82
Grayling	1	32	6	1	0	0	1	40	.80	
	2	2	40	1	0	0	0	43	.93	
	3	0	0	0	0	0	0	0	-	
	4	0	0	0	0	0	0	0	-	
	5	0	0	0	0	0	0	0	-	
	6	0	0	1	0	0	6	7	.86	
										78/90 = .87
Yuma	1	150	3	2	1	0	0	156	.96	
	2	2	45	4	0	7	0	58	.78	
	3	0	0	0	0	0	0	0	-	
	4	0	0	0	0	0	0	0	-	
	5	0	3	0	0	19	0	22	.86	
	6	1	0	0	0	0	2	3	.67	
										216/239 = .90

Overall Accuracy: 390/446 = .87

The site independence measure, S, was obtained as follows. The six class accuracies were computed for each of the three sites. The value of S was set equal to the ratio of the lowest site accuracy to the best site accuracy. Thus, if a classifier performed well at one or more sites, but poorly at one of the other sites, S was small. Conversely, S approached 1 as a given classifier produced consistent, i.e., the same, accuracy at all sites.

The overall performance measure was computed as the product of the three criteria of success:

$$P = A \times C \times S$$

Good performance exists when A, C, and S each approach 1, as does P. Therefore, any group of classifiers can be rated on the above performance scale, which ranges from 0 for poor performance to 1 for perfect performance.

Notice that a classifier that had an overall accuracy of 90 percent (A = 0.9), with a standard deviation of 10 percent (C = 1 - 0.1 = 0.9), and a worst-site-to-best-site accuracy ratio of 90/100 (S = 0.9/1.0 = 0.9) -- all very good values -- would achieve a performance value of

$$P = 0.9 \times 0.9 \times 0.9 = 0.729.$$

Thus, in practice, a P value greater than about 0.7 can be viewed as signifying excellent performance.

The performance of the APN classifier using both decision logics can be obtained from Tables 5 and 6:

	A	C	S	P=ACS
Tie-Breaking Logic:	.845	.958	.870	.704
Vote Reporting Logic:	.874	.946	.908	.751

Therefore, both voting logics produce good performance measures (P > 0.7), with the vote reporting logic (V=4) giving slightly better results.

Summary of Conclusions

It has been demonstrated that a seismic/acoustic six-way target classifier can be realized that, in simulations, exhibits improved overall accuracy, improved site independence and improved class invariance, and improved range invariance. The specific conclusions reached are:

1. An average single epoch classification accuracy of 85 percent can be realized with a practicable design.
2. The above accuracy is achieved with high consistency at different sites and for the different target classes, and the classifier is relatively insensitive to target range (out to the periphery of the target detection zone) and the speed, altitude (where applicable), and heading of the target.
3. By utilizing a pairwise voting logic structure, the classifier circuitry is potentially less prone to manufacturing tolerance errors and to parameter drift.
4. Using vote reporting in lieu of class reporting, the voting structure is also suitable for multi-target classifications. Furthermore, the likelihood of intentional or unintentional jamming of the sensor is reduced, and the user has greater opportunity to exercise judgment concerning the tactical situation.
5. Further work is needed to develop the most cost-effective classifier design. As a foundation for this work, additional field and/or synthetic data should be obtained so as to represent more fully the wide variety of targets and terrain conditions that could be encountered by an operational system.

Acknowledgments

This work was supported by the U. S. Army Mobility Equipment Research and Development Center (MERDC) under Contract No. DAAK02-74-C-0322. The authors thank Dr. R. K. Young of MERDC for his help and guidance throughout the project.

References

1. Barron, R. L., "Learning Networks Improve Computer-Aided Prediction and Control," August 1975, Computer Design.
2. Hunt, S. P., M. D. Layman, and D. L. Wilson, Seismic Acoustic Target Classifier, GTE Sylvania, Inc. FTR to USA MERDC, Contract DAAK02-72-C-0546, June 1974.
3. Mucciardi, A. N., "Elements of Learning Control Systems With Applications to Industrial Processes," Proc. 1972 IEEE Conference on Decision and Control, New Orleans, La. December 13-15, 1972, pp. 320-325.

4. Roth, R. R., Design and Development of the Seismic Target Classifier, Honeywell, Inc. FTR to USA MERDC, Contract DAAK02-72-C-0547, July 1973.

5. Roth, R. R., Seismic Acoustic Target Classifier, Honeywell, Inc. FTR to USA MERDC, Contract DAAK02-72-C-0547.

6. Scott, R. W. and M. D. Laymon, Seismic Target Classifier, GTE Sylvania, Inc. FTR to USA MERDC, Contract DAAK02-72-C-0546, December 1973.

7. Shankar, R., A. N. Mucciardi and E. E. Gose, "Classification into K-Categories Via Discrimination Between Pairs of Categories," 1975 Proc. Milwaukee Symposium on Automatic Computation and Control, Milwaukee, Wisconsin, April 17-19, 1975.

8. Whalen, M. F. and A. N. Mucciardi, Synthesis of Nonlinear Adaptive Learning Network Seismic Target Classifier, Adaptronics, Inc. Interim Report to USA MERDC, Contract DAAK02-74-C-0322, March 1975.

9. Whalen, M.F., J. D. Sanders and A. N. Mucciardi, Performance Evaluation of Nine Candidate REMBASS Single-Target Classifiers, Adaptronics, Inc. Final Technical Report to USA MERDC, Contract DAAK02-74-C-0322, February 1976.

APPENDIX H

CMOS/SOS SERIAL-PARALLEL MULTIPLIER

D. Hampel, K. E. McGuire,
an K. J. Prost

CMOS/SOS Serial-Parallel Multiplier

DANIEL HAMPEL, SENIOR MEMBER, IEEE, KENYON E. McGUIRE, MEMBER, IEEE, AND KALMAN J. PROST

Abstract—A 24-bit serial-parallel multiplier was integrated in CMOS/silicon-on-sapphire (SOS) technology on a 155 mil × 170 mil chip. The operation of this multiplier is described, showing how the parallel loaded multiplier x combines with the serial loaded multiplicand, a, to form the serial product. An addend, b, can also be accommodated to produce $ax + b$. The design of the multiplier cells are based on functional majority logic adders and weak or trickle inverter master-slave latches. The chip operates at clock rates up to 18 MHz. Power dissipation at 10 MHz and V_{DD} of 5 V is about 20 mW, and the energy consumption for multiplying two 16-bit numbers is about 64 nJ. Typical application areas are mentioned.

Manuscript received April 14, 1975; revised June 6, 1975. This work was supported by the Air Force Avionics Laboratory under Contract F33615-73-1089, "LSI Electronically Programmable Arrays" under the guidance of C. Gwinn.

The authors are with the RCA Government Communications and Automated Systems Division, Somerville, N.J.

INTRODUCTION

ADVANCES in digital signal processing are being made possible by improved large-scale integrated (LSI) multipliers. Such multipliers have been built in a variety of bipolar and MOS technologies for use in different types of processors. Basically, two types of multiplier architectures are of interest for high performance; all parallel [1], [2] and serial-parallel [3]-[5]. The all-parallel multiplier has, as a minimum, a total of N^2 adders, as well as N^2 partial product gates for an $N \times N$ capacity. The serial-parallel multiplier has N adders and two N latches (for storing and shifting) to successively accumulate the outputs of N partial product gates. The multiplicand must be available in serial while the multiplier is fed in parallel; the product output is also serial. The serial-parallel multiplier is thus much simpler to implement (as will

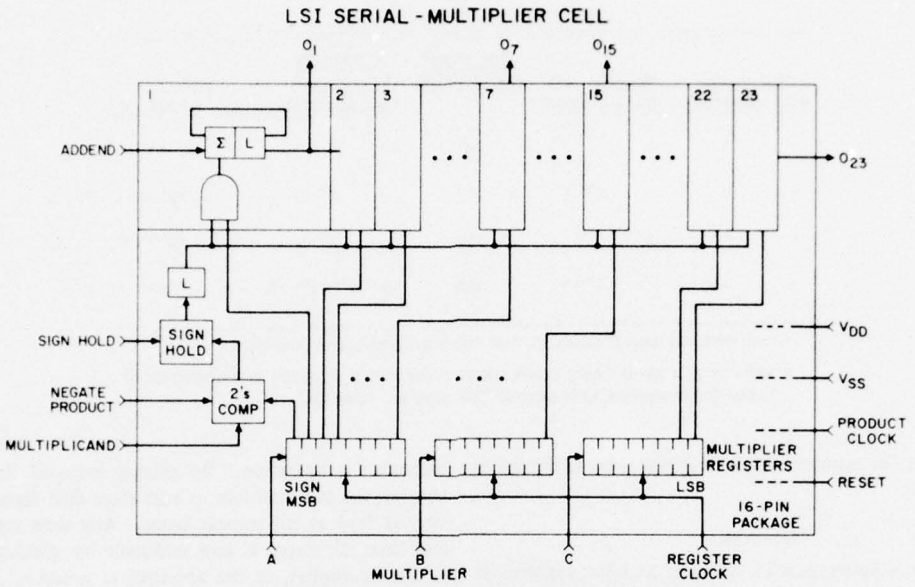


Fig. 1. LSI serial–parallel multiplier functional diagram.

be seen) and requires fewer connections. However, the speed for a multiplication of two N-bit numbers is $2N \times 1/f$, where f is the control clock frequency governing the multiplicand input rate and output product. The speed of the parallel array multiplier, on the other hand, is generally dependent on the asynchronous ripple through time of the adder net, and for a given technology is substantially faster than the serial–parallel multiplier. For many types of digital signal processors, where many multiply operations can be done simultaneously, the lower speed single multiply of the serial–parallel approach can be compensated for. This is done by using word-parallel, bit-serial processing, employing several multipliers. Example of processors which can optimally use much multipliers include fast Fourier transform (FFT) arithmetic units [4], digital filters [5], and programmable arrays used for complex, nonlinear transformations [6] for classification, modeling, and control.

The serial nature of the multiplier to be described using a predominance of on-chip processing and the desire to pack a 24-bit array on a single chip all lead to selecting CMOS/silicon-on-sapphire (SOS) as an ideal technology choice for implementing the multiplier. This paper will describe the logic and circuit design of the multiplier, give performance results of samples of fabricated units, and suggest applications.

Multiplier Description

Fig. 1 is the functional block diagram of the multiplier that was integrated. The module handles the multiplicand in two's complement form, serially fed, and the multiplier in sign-magnitude form, with a sign bit and up to 23 significant bits stored in parallel. Modules may be cascaded to form higher order terms or to handle larger multipliers.

The module contains the following.

1) A 24-bit (23 bits plus sign) holding register for the multiplier input. This register is organized as three 8-bit serial-in, parallel-out registers as a compromise between the number of multiplier input connections and time required to load the register. A separate clock controls loading of the multiplier register.

2) 23 partial product gates (AND gates) and 23 adder–latch ($\Sigma - L$) stages which actually perform the multiplication by either adding the contents of the multiplier register and shifting the contents to the right or by shifting right only, depending on the multiplicand input. By providing access to the input of the first adder, an addend can be fed in, synchronously with the multiplicand, effectively giving the multiplier the capability to perform an $ax + b$ calculation.

The outputs of stages 1, 7, 15, and 23 are made available so that the cell may be used with 8-, 16-, or 24-bit multipliers (the additional bit being the sign), as well as a 1-bit adder.

The serially fed multiplicand and addend can be of any desired length.

3) Control logic providing the following functions. The "two's complement" flip-flop and an associated EXCLUSIVE-OR gate (not shown) will two's complement the multiplicand if the multiplier sign bit is a 1. This operation results in a product which is in two's complement form, as shown in Table I.

The "sign hold" flip-flop serves two functions: it separates the propagation delay of the two's complementer circuit from that of the adder–latch input stages, and it provides the sign hold function. Propagation of the multiplicand through the two's complement flip-flop and its associated logic increases the multiplicand input delay to the adder/latches. This increased delay would limit the overall speed of the multiplier by reducing the multiplicand throughput rate. By splitting up the multiplicand input delays, higher multiplicand throughput

TABLE I
TWO'S COMPLEMENTER OPERATION

MULTIPLIER		MULTIPLICAND		TWO'S COMP MULTIPLICAND	CORRECTED MULTIPLICAND		PRODUCT	
SIGN	MAGNITUDE	SIGN	MAGNITUDE		SIGN	MAGNITUDE	SIGN[1]	MAGNITUDE[2]
+	X	+	Y	NO	+	Y	+	XY
+	X	−	2^N-Y	NO	−	2^N-Y	−	$X2^N-XY$
−	X	+	Y	YES	−	2^N-Y	−	$X2^N-XY$
−	X	−	2^N-Y	YES	+	$2^N-(2^N-Y)$ $=Y$	+	XY

1. THE PRODUCT SIGN IS EQUAL TO THE CORRECTED MULTIPLICAND SIGN.
2. $X2^N-XY$ IS A VALID TWO'S COMPLEMENT FORM IF THE PRODUCT IS TRUNCATED TO ELIMINATE OVERFLOW BITS BEYOND THE SIGN BIT POSITION.

is achieved at the expense of a 1-bit initial delay in the multiplicand input.

Operation

First, the multiplier must be stored in its input register. If an addend, b, is to be added to the product (ax), its least significant bits (LSB) must be loaded before the multiplicand is entered.

Since bits of equal significance must be added together, the first 22 bits of the addend must be shifted into the cell before multiplication starts. This preshifting will cause the addend LSB (now at the output of cell 22) to be added to the product LSB in the 23rd adder-latch stage. When multiplication starts, the addend input will correspond to the 2^{22} bit in the addend. Then the multiplication begins.

When the multiplicand sign bit has been entered into the cell, the 24 LSB's of the product have been clocked out of output O_{23}. Since the product of two 23-bit plus sign numbers contains 46 bits plus sign, the remaining 22 significant bits and the new sign bit information are still within the 23 cell stages. This information cannot be merely shifted out, as carries from previous additions must be allowed to modify this information.

This problem can be solved by making the multiplicand 47-bits long. Leading nonsignificant bits can be inserted between the multiplicand sign bit and the most significant bit (MSB). Since the multiplicand is in two's complement form, leading nonsignificant bits are always identical to the sign bit. By "stretching" the sign bit 23 additional places, the effect of lengthening the multiplicand to 47 bits is achieved. In this way all of the product bits are shifted out of O_{23}. As indicated in Table I, overflow bits beyond the sign must be eliminated to preserve the two's complement form of the output. Sign-bit stretching is accomplished by disabling the clock input to the sign hold flip-flop when the sign bit is in that flip-flop.

The output of this flip-flop becomes an "enable" signal which will allow the multiplier bits to be added to the adder-latch contents when this signal is a 1. The shift-and-add/shift only algorithm mentioned earlier is thus implemented.

The module can perform other functions in addition to the basic $ax + b$ operation. By placing zeros in the multiplier register, the cell functions as a 23-stage shift register with the addend lead as the register input. Any shift register length less than 23 stages is also realizable by placing a single 1 in the multiplier, in the appropriate position. By placing 01000000000000000000000 in the multiplier register, only Stage 1 will function as an adder, while the other stages merely shift, causing the module to function as a serial adder and register. Placing a 1 on the "negate product" lead results in a serial subtractor and register.

Circuit Design

The main area consumption factors of the cell are the adders and latches used for the 23 stages of the multiplier.

Fig. 2(a) is the block diagram for the repetitive portion of the multiplier representing the bulk of the chip. All latches are based on the trickle inverter master-slave sections shown in Fig. 2(b). This includes the multiplier register, and sum and carry latches. Only the sum latch has a parallel reset capability. The partial product AND gate is realized with a transmission gate, shown feeding the Y input to the full adder. The design of the adder and latches will be described.

The trickle inverter register cell combines the simplicity of the dynamic register cell with the advantages of full static operation. A special inverter is designed with a greatly reduced output drive capability so as not to interact with the input signal. Even with a transconductance 24 times less than the normal inverter, this trickle inverter still provides more than 10 times the current required to cancel the highest leakage currents observed in SOS-CMOS circuits, e.g.,

maximum output current of trickle inverter = 86 μA at 10 V
maximum observed leakage = 8 μA/gate.

The trickle inverter will hold the logic state indefinitely as did the conventional static register cell. When new data are read into the cell, the trickle inverter is overdriven by a source conductance at least 5 times greater than the trickle inverter output so as to avoid significant increases in stage delay. Advantages of the trickle inverter cell over a conventional static register cell include reduced chip size, reduced loading on

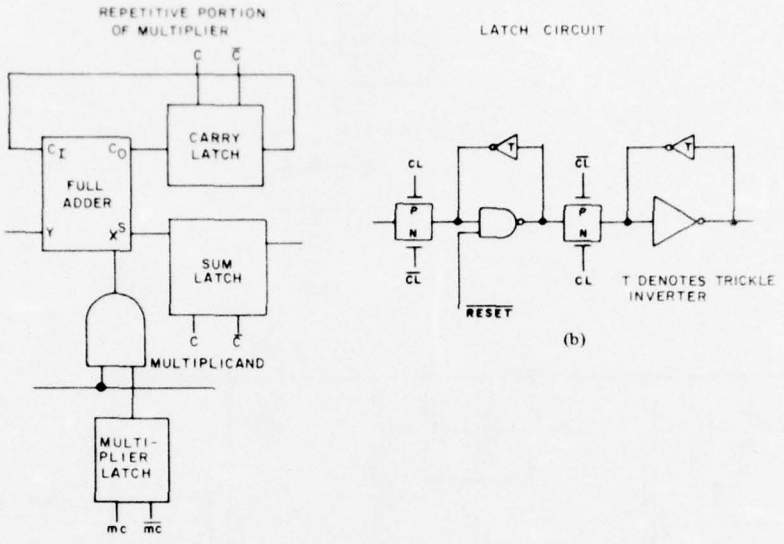


Fig. 2. (a) Repetitive portion of multiplier. (b) Latch circuit.

clock lines, and less sensitivity of this circuit to effects of clock skewing.

The logic and circuit schematic for the full adder shown in Fig. 3 uses two functional gates to provide $\overline{C_{out}}$ and $\overline{Sum}$, each realizing the reduced majority function of their inputs. The first gate produces $\overline{C_{out}}$ (low) as two or more out of the three inputs are low. This signal ($\overline{C_{out}}$) feeds the second gate with a double-weighted value along with the three input signals. The sum (high) output then reflects whether three out of the five (or more) weighted inputs are high. The major advantage of this adder is its minimized transistor requirements. Total transistors used, including inverters used for driving as well polarity reversal, is 28. The inverters have twice the channel width as the gate devices. Simulations of this circuit have indicated an approximate delay of 10 ns for carry-out and 20 ns for sum-out.

Performance

The integrated circuit is shown in Fig. 4. The initial samples were fabricated by a deep depletion process, yielding devices with a V_T of 2 V for P devices and a V_T of 1 V for N devices. A special test and control unit was constructed, employing Schottky TTL parts, to load and exercise the multiplier. Also, the chip multiplier logic was duplicated in Schottky TTL for use as a reference for test and evaluation of the fabricated chips. The TTL version required 35 IC's, dissipating 7.5 W. A complete set of test waveforms demonstrating the operation of the chip is shown in Fig. 5, in performing an $ax + b$ calculation. In the example shown, a is the 24-bit multiplier, preloaded as three 8-bit words; X is the 24-bit multiplicand and b is the 48-bit addend. The wafer probe test photos were made at a V_{DD} supply of 5 V and a clock rate of about 1 MHz. The values in the example, in octal code, are

$$a = 03413405$$

$$x = 07403416$$

$$b = 060004241400105$$

output $ax + b = 112347040462613$.

The results of tests run on packaged units to determine power dissipation at various supply voltages are given in Fig. 6. These curves include dc power due to leakage as well as dynamic power dissipation. The data represent the average of six samples.

The multiplier efficiency or, the amount of energy it expends in performing a given calculation, was determined. Since it takes $2n$ clock pulses to obtain the complete product of two n-bit numbers, the energy, E, per calculation is

$$E = P_D \times \frac{2n}{F}$$

where P_D is the chip's power dissipation at F and F is the clock frequency.

At a V_{DD} of 5 V, and a clock rate of 10 MHz, the energy consumption for multiplying two 16-bit numbers is

$$E = 20 \times 10^{-3}\ W \times \frac{32\ S}{10^7} = 64\ \text{nJ}.$$

This energy is fairly constant for a given operating voltage, at different clock frequencies, since the power dissipation is proportional to the clock frequency. By comparison, one other LSI chip containing a 16 X 16 or greater multiply capacity [4]

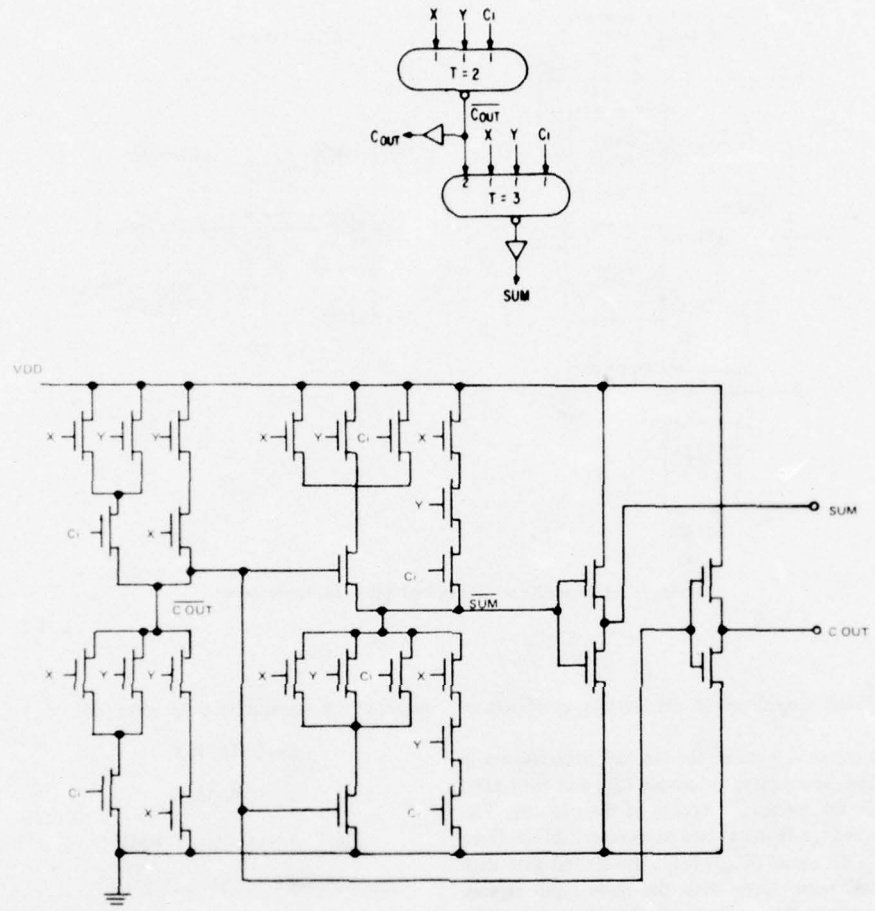


Fig. 3. Full adder used in multiplier.

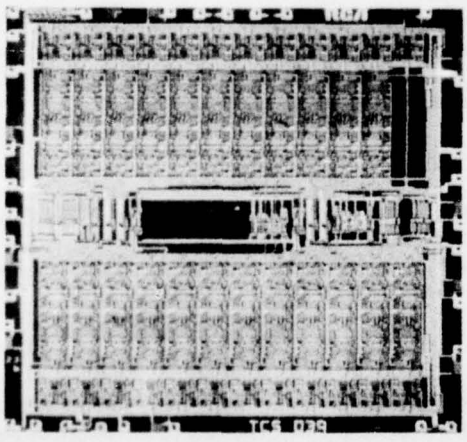


Fig. 4. Photomicrograph of CMOS/SOS multiplier (155 mils × 170 mils).

is a high-speed triple-diffused bipolar device producing an output in 330 ns while dissipating 3.5 W, for an energy of 100 nJ.

Applications

The multiplier described is most advantageously used in those applications where many simultaneous multiply operations are required, or can be implemented. In such situations, word-parallel, bit-serial architectures are used. Since there are up to 24 bits of multiplier capacity per chip, system efficiency is increased when many multiplier chips are effectively enployed. Such architectures amortize the control logic necessary to exercise the multiplier, i.e., providing appropriate clock signals for the chip(s), etc. For example, in their use for FFT units, higher throughputs would be possible using radix-4 organizations, employing more multiplier parallelism. In their use for multinomial realizations, such as in the evaluation of the following expression,

$$y = w_0 + w_1 X_1 + w_2 X_2 + w_3 X_1 X_2 + w_4 X_1^2 + w_5 X_2^2,$$

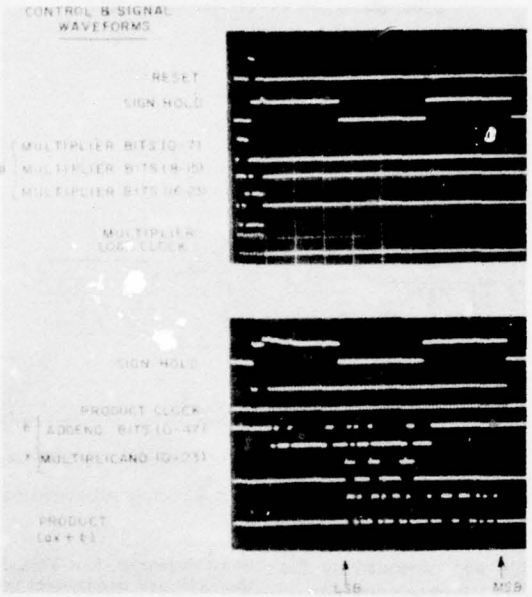


Fig. 5. Multiplier control and signal waveforms.

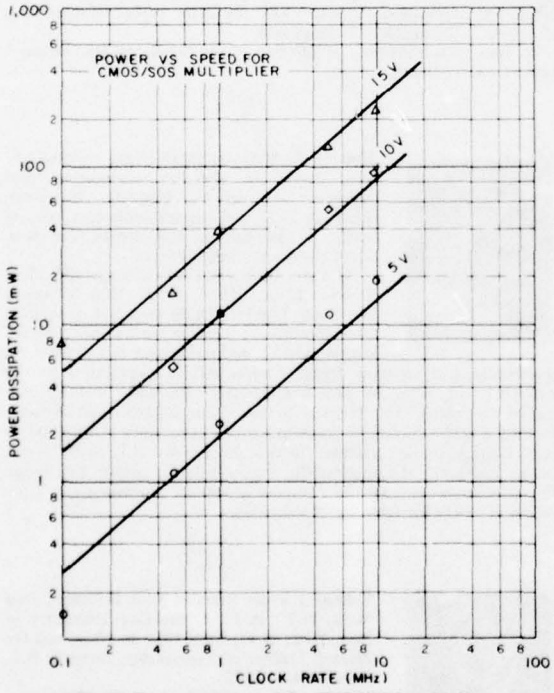


Fig. 6. Power versus speed for CMOS/SOS multiplier.

a serial-parallel multiplier bank is employed as shown in Fig. 7. All first-order multiplications are done simultaneously, while second-order multiplications are combined with the accumulation feature of the cell ($ax + b$) so as to form the output function most efficiently.

This multiplier can also be used with a microprocessor to provide the micro with a hardware multiply feature. In particular, the serial-parallel multiplier, operating from a separate 10-MHz clock with control logic supervised by the microprocessor, can be shown to give the COSMAC[1] a 35 μs multiply time for multiplying two 16-bit numbers. Much of this time is in fact due to the data transfer to and from the processor and multiplier. This represents about a 100 to 1 improvement in speed compared to software multiply.

Summary and Conclusions

A well known multiplier architecture has been implemented in CMOS/SOS for improved performance. Basically an $ax + b$ module, this multiplier was integrated on a 155 mil X 170 mil chip with 24-bit capacity. The relatively small chip size and low pin count required (16 pins) for this type of multiplier will lead to more economical signal processing implementations or to the incorporation of more associated logic on the same chip where higher performance is necessary. Although the maximum clock rate of the initial design is about 18 MHz, it is felt that modifications can be made to increase this to 30 MHz. These modifications would include a faster adder, where the sum output gate does not have to wait for carry output generation (at a slight increase in component count), and improved layout to minimize RC time constants associated with interconnect tunnels.

Acknowledgment

The efforts of R. Blasco, B. Altschuler, and R. Stewart are acknowledged for aid in logic design, simulation, and circuit

[1] COSMAC is an RCA CMOS two chip 8-bit microprocessor [7].

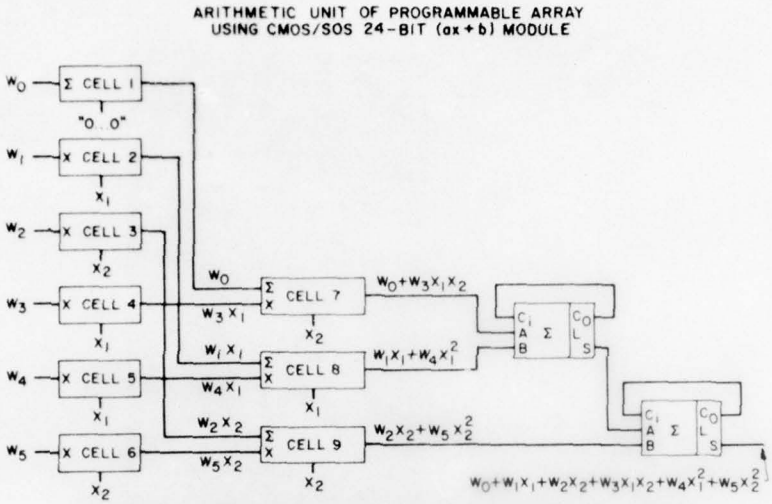


Fig. 7. Word parallel processor using custom multiplier for polynomial evaluation.

layout of the multiplier. The chip was fabricated by the Solid-State Technology Center, under H. Borkan's direction.

References

[1] S. D. Pezaris, "A 40 ns 17-bit by 17-bit array multiplier," *IEEE Trans. Comput.* (Short Notes), vol. C-20, pp. 442–447, Apr. 1971.

[2] G. W. McIver, R. W. Miller, and T. G. O'Shaughnessy, "A monolithic 16 × 16 digital multiplier," in *1974 Int. Solid-State Circuits Conf., Dig. Tech. Papers*, pp. 54–55.

[3] L. B. Jackson, J. F. Kaiser, and H. S. McDonald, "An approach to the implementation of digital filters," *IEEE Trans. Audio Electroacoust.* (*Special Issue on Digital Filters: The Promise of LSI Applied to Signal Processing*), vol. AU-16, pp. 413–421, Sept. 1968.

[4] N. R. Powell and J. M. Irwin, "A MOS monolithic chip for high-speed flexible FFT microprocessors," in *1975 Int. Solid-State Circuits Conf., Dig. Tech. Papers*, pp. 18–19.

[5] G. P. Edwards, P. J. Jennings, and T. Preston, "A MOS LSI double second order digital filter circuit," in *1975 Int. Solid-State Circuits Conf., Dig. Tech. Papers*, pp. 20–21.

[6] D. Hampel, R. W. Blasco, and D. Cleveland, "Electronically programmable LSI arrays," in *Conv. Rec., 1974 NAECON*, pp. 134–141.

[7] N. P. Swales and J. A. Weisbecker, "COSMAC—A microprocessor for minimum cost systems," in *1974 IEEE Intercon, Dig. Tech. Papers*.

Daniel Hampel (M'64-SM'71) received the B.S.E.E. and M.S.E.E. degrees from the Newark College of Engineering, Newark, N.J., in 1953 and 1958, respectively. In addition, he graduated from the Communications Development Training Program, Bell Laboratories, Murray Hill, N.J., in 1956.

From 1953 to 1957, he designed and developed digital code translation, selection, and amplification circuits for electronic switching systems at Bell Laboratories. As a Senior Engineer with the ITT Corporation from 1957 to 1960, he worked on special-purpose data-processing systems to perform analyses of communications data. He joined RCA in 1960, and until 1962 was a Project Engineer at the Nuclear and Scientific Service Department where he designed computing circuits for data-handling systems for nuclear experiments. He joined RCA's Communications Systems Division Laboratories, New York, N.Y., 1962. He has been in charge of threshold logic development and logic design in support of divisional requirements. He has recently been appointed Manager of the Advanced Communications Laboratory, Somerville, N.J. His present responsibilities include signal processing techniques development and design and application of custom LSI. He is the author of more than 10 papers in the digital techniques area.

Mr. Hampel is a member of Tau Beta Pi and Eta Kappa Nu. He has 3 patents and 2 pending.

Kenyon E. McGuire (S'67-M'69) received the B.S. degree in electrical engineering from Worcester Polytechnic Institute, Worcester, Mass., in 1969, and has accumulated all course credits toward the M.S.E.E. degree from New York University, New York, N.Y.

He was a Design Engineer at Singer–Kearfott Division from 1969 to 1973. Here he was in the Radar Support Equipment and Automatic Test Equipment Groups where he designed digital, analog, and microwave test equipment, specifically, high-accuracy digital Doppler spectrum simulators for the AN/APM 153, AN/APN 185, and AN/APN 190 and computer controlled test equipment currently in use at Singer–Telesignal Division. He joined RCA's Advanced Communications Laboratory of the Government Communication Systems Division, Somerville, N.J., in 1973, as a Senior Engineer. His responsibilities have included custom LSI design for programmable arrays for advanced numerical processing and application of CCD's for video signal processing.

Kalman J. Prost attended RCA Institutes, New York, N.Y., in 1958, the City University of New York, N.Y., from 1958 to 1962, and the Newark College of Engineering, Newark, N.J., from 1968 to 1972.

From 1963 to 1965 he was with IBM Corporation in the Test Equipment Engineering Department. Here he wrote test procedures for automatic checkout of digital circuits, and initiated changes for existing equipment. From 1958 to 1963 he worked for the Arma Corporation. During this time he worked on system test, quality control, and field engineering on the Inertial Guidance System for the Atlas

Missile. He joined the RCA Corporation in 1966, and was assigned to the Advanced Digital Techniques Group of Government Communications Systems New York, N.Y., and presently, Somerville, N.J. Since joining the group he has been involved in work on integrated threshold gates, logic design, and radiation hardening. In these areas he has assumed responsibility for design and testing. As a Senior Engineer, he has recently designed and tested the digital portion of a MODEM for spread spectrum communication, including the application of CMOS/SOS correlators for PRN code synchronization. He has also designed and evaluated a Viterbi decoder for use in signal-to-noise improvement at low bit-error rates and is now designing a TDM switch. He has co-authored three papers in the digital techniques area.

APPENDIX I

I.1 - Internal Representation of Data in the EAI-640 Computer

I.2 - EAI-640 Basic Instruction Repertoire

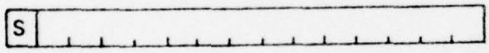
I.3 - Simulation Computer Programs

DATA FORMAT

A single datum consists of one or more words of information depending on whether the mode is integer, logical, real, double-precision, or complex. Negative mantissas and/or exponents are carried in two's complement form. The high order word of all data is carried in the (A) register during computation where it is expected by the code generated for the logical and relational IF statement. The exact formats for the different modes are as follows:

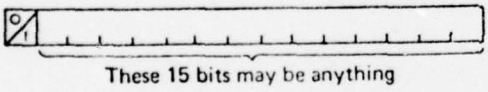
INTEGER

An integer datum consists of a right-justified 15-bit (plus sign) integer value carried in a single word.



LOGICAL

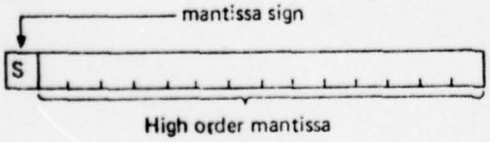
A logical datum consists of a one-bit logical value; 1 means .TRUE. and 0 means .FALSE.. This bit is carried in the sign portion of the register.



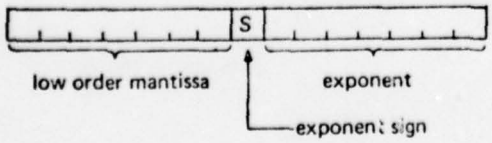
REAL

A real datum consists of normalized 23-bit (plus sign) mantissa along with a 7-bit (plus sign) integer exponent carried in two consecutive words. A zero has a zero mantissa but the exponent equals '200.

1st word



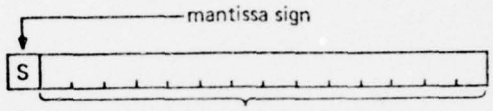
2nd word



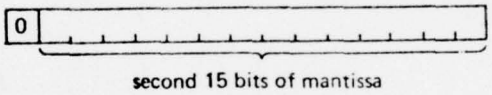
DOUBLE-PRECISION

A double-precision datum consists of a normalized 53-bit (plus sign) mantissa along with a 7-bit (plus sign) integer exponent carried in four consecutive words. A zero has exponent = '200.

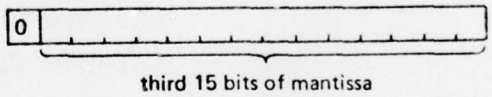
1st word



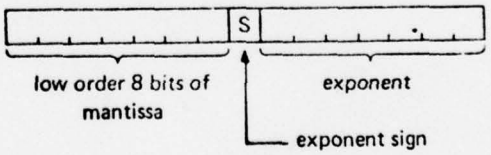
2nd word



3rd word



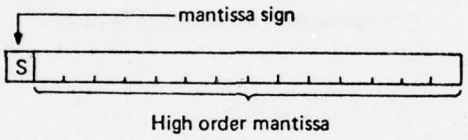
4th word



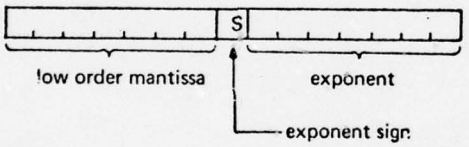
COMPLEX

A complex datum consists of two real data carried in four consecutive words. The first real datum contains the real part of the complex datum while the second real datum carries the imaginary part.

1st word*

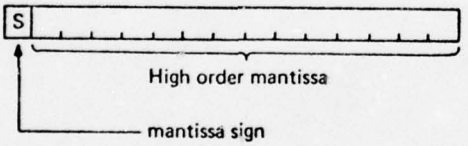


2nd word*

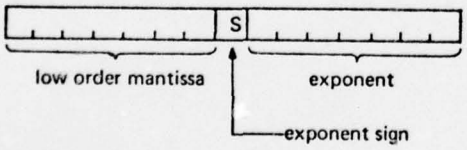


*real part

3rd word*



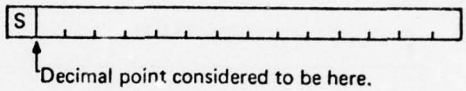
4th word*



*imaginary part

SCALED FRACTION

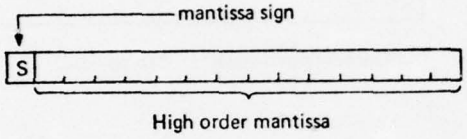
A scaled fraction datum consists of an unnormalized fractional value carried in a single word. The decimal point is considered to be placed between the sign bit and bit 1.



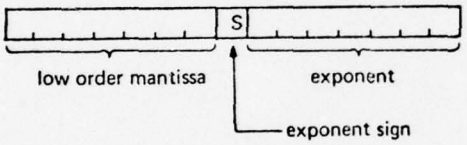
COMPLEX

A complex datum consists of two real data carried in four consecutive words. The first real datum contains the real part of the complex datum while the second real datum carries the imaginary part.

1st word*

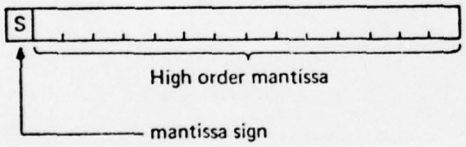


2nd word*

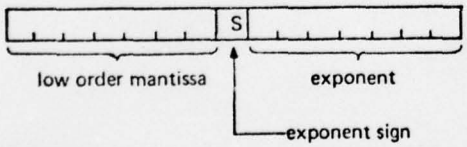


*real part

3rd word*



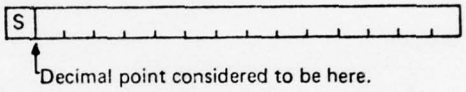
4th word*



*imaginary part

SCALED FRACTION

A scaled fraction datum consists of an unnormalized fractional value carried in a single word. The decimal point is considered to be placed between the sign bit and bit 1.



FUNCTIONAL LISTING OF 640 INSTRUCTIONS

MNEMONIC CODE	OCTAL CODE	INSTRUCTION DESCRIPTION	TIME IN MICROSECONDS	CONDITION CODE**
TRANSFERS				
LA	14EDDD	Load Accumulator	3.30	U
STA	16EDDD	Store Accumulator	3.30	U
LX	05(E+2)DDD*	Load Index Register	3.30	U
STX	05EDDD*	Store Index Register	3.30	U
ARITHMETIC				
A	15EDDD	Add	3.30	1
S	17EDDD	Subtract	3.30	1
M	03EDDD	Multiply	18.15	2
D	01EDDD	Divide	18.975	2
SQR	021400	Square Root	16.5	2
AOA	020040	Add One to Accumulator	1.65	1
AOM	07EDDD	Add One to Memory and Skip	4.95	U
TCA	020100	Two's Complement Accumulator	1.65	1
CLR	026740	Clear Accumulator	1.65	U
CAO	020000	Clear and Add One to Accumulator	1.65	1
LOGICAL				
OR	10EDDD	Or (Logical Sum)	3.30	3
XOR	11EDDD	Exclusive Or (Logical Subtract)	3.30	3
AND	13EDDD	And (Logical Product)	3.30	3
C	12EDDD	Compare	3.30	4
OCA	020200	One's Complement Accumulator	1.65	U
SHIFTS				
ARS	0260(40+SH)	Arithmetic Right Single	See Note (1) Below	U
ALS	0260SN	Arithmetic Left Single		5
ARD	0261(40+SH)	Arithmetic Right Double		U
ALD	0261SN	Arithmetic Left Double		5
LRS	0262(40+SH)	Logical Right Single		U
LLS	0262SN	Logical Left Single		U
LRD	0263(40+SH)	Logical Right Double		U
LLD	0263SN	Logical Left Double		U
CONTROL				
SMP	024440	Set Multiple Precision Bit, Reset Carry Borrow Bit	1.65	U
RMP	024400	Reset Multiple Precision Bit	1.65	U
SSP	026400	Set sign of Accumulator Positive	1.65	U
SSN	026440	Set sign of Accumulator Negative	1.65	U
NOP	027400	No Operation	1.65	U
P	025***	Pause	1.65	U
T	027***	Trap	1.65	U
INTERRUPT				
SMI	024540	Set Master Interrupt Bit	1.65	U
RMI	024500	Reset Master Interrupt Bit	1.65	U
PROGRAM PROTECT				
SPB	020440	Set Protect Bit	3.30	U
RPB	020400	Reset Protect Bit	3.30	U
EXCHANGES				
EX	026500	Exchange Accumulator and Index Register	1.65	U
EQ	026540	Exchange Accumulator and Q Register	1.65	U
EP	026600	Exchange Accumulator and Program Counter	1.65	U
ES	026700	Exchange Accumulator and Program Status Word	1.65	CC=A14 & 15
JUMPS & SKIPS				
J	04EDDD	Jump Unconditional	1.65	U
L	06EDDD	Link	3.30	U
SSW	0234+SW	Skip on Sense Switch	2.475	U
		SW A = 200_8 F = 10_8		
		B = 100_8 F = 4_8		
		C = 40_8 G = 2_8		
		D = 20_8 H = 1_8		
SKU	027417	Skip Unconditional	2.475	U

*E = 0, 1, 4 or 5 only. **U = Condition Code unchanged; 1, 2, 3, 4, 5 = valid class of Skip on Condition Code instructions. ***0 to 377_8

EDDD = Four Octal digits representing Effective Address option and Displacement Address

SH = Shift Count

Note (1):
Odd: $\left(\frac{N}{7}\times 1.65\right)+1.65$ = number of microseconds
Even: $\left(\frac{N}{7}\times 1.65\right)+.75$ = number of microseconds
Where N = number of positions shifted.

FUNCTIONAL LISTING OF 640 INSTRUCTIONS (Continued)

MNEMONIC CODE	OCTAL CODE	INSTRUCTION DESCRIPTION	TIME IN MICROSECONDS	CONDITION CODE**
SKIP ON REGISTER CONDITION				
ICX	022XXX+	Increment Index and Skip	2.475	U
DCX	022XXX−	Decrement Index and Skip	2.475	U
SKN	024000	Skip if Accumulator Negative	1.65	U
SKP	024040	Skip if Accumulator Positive	1.65	U
SAE	024100	Skip if Accumulator Even	1.65	U
SQE	024140	Skip if Q Register Even	1.65	U
		XXX+ positive number 1 to 377_8		
		XXX− negative number − 1 to -400_8		
SKIP ON CONDITION CODE				
Class 1: Valid following A, S, AOA, TCA				
SZ	027410	Skip if result was Zero	2.475	U
SP	027404	Skip if result was Plus	2.475	U
SM	027402	Skip if result was Minus	2.475	U
SO	027401	Skip if result caused Overflow	2.475	U
SNZ	027407	Skip if result was Not Zero	2.475	U
SNP	027413	Skip if result was Not Plus	2.475	U
SNM	027415	Skip if result was Not Minus	2.475	U
SNO	027416	Skip if result did Not cause Overflow	2.475	U
SPZ	027414	Skip if result was Plus or Zero	2.475	U
SMZ	027412	Skip if result was Minus or Zero	2.475	U
SZO	027411	Skip if result was Zero or caused Overflow	2.475	U
SPM	027406	Skip if result was Plus or Minus	2.475	U
SPO	027405	Skip if result was Plus or caused Overflow	2.475	U
SMO	027403	Skip if result was Minus or caused Overflow	2.475	U
Class 2: Valid following M, D and SQR				
SO	027401	Skip if result caused Overflow	2.475	U
SNO	027416	Skip if result did Not Cause Overflow	2.475	U
Class 3: Valid following OR, XOR and AND				
SZ	027410	Skip if result was Zero	2.475	U
SNZ	027407	Skip if result was not Zero	2.475	U
Class 4: Valid following C				
SE	027410	Skip if Operands Equal	2.475	U
SG	027402	Skip if Accumulator was Greater	2.475	U
SL	027404	Skip if Accumulator was Less	2.475	U
SNE	027406	Skip if Accumulator was Not Equal	2.475	U
SGE	027412	Skip if Accumulator was Greater or Equal	2.475	U
SLE	027414	Skip if Accumulator was Less or Equal	2.475	U
Class 5: Valid following ALS and ALD				
SO	027401	Skip if result caused Overflow	2.475	U
SNO	027402	Skip if result did Not cause Overflow	2.475	U
SAO	027404	Skip if result is About to Overflow	2.475	U
NAO	027410	Skip if result is Not About to Overflow	2.475	U
INPUT/OUTPUT				
DI	002DDN	Data Input	3.30	U
DO	003DDN	Data Output	3.30	U
RI	000PDN	Record Input	6.6+1.65 per word	U
RO	001PDN	Record Output	6.6+1.65 per word	U
DF	005DDN	Device Function	3.30	U
SI	004DDN	Status Input	3.30	U
TTI	006T00	Timer Channel	(not pending) 2.47	U
TDI		Test Device Interrupt	(pending) 7.42	
	0066DN	Channel Zero		
	0067DN	Channel One		
	0070DN	Channel Two or DMAC Devices		
	0071DN	Channel Three including Teletype		
	007137DN	Direct Memory Access Controller		
TAI	0072DN	Alarm Channel		
BRI	000(4+P)DN	Buffered Device Input	6.6+1.65 per word	U
BRO	002(4+P)DN	Buffered Device Output	6.6+1.65 per word	U
BDF	005(4+P)DN	Buffered Device Function	3.30	U
BSI	004(4+P)DN	Buffered Status Input	3.30	U

DN = Device Number in octal
P = Record packet number
T = Timer number

PAGE 1 C FPNET

```
C THIS PROGRAM IS A FIXED POINT VERSION OF PNET DESIGNED TO WORK
C ON THE 640 USING THE SCALED FRACTION FIXED POINT WORD FORMAT.
C
C
      DIMENSION X(50),W(15,8),IFM(40),ALPHA(50),N1(50),N2(50),NW(50)
     1,   SCALE(16),YSAVE(8),EABS(50)
      DOUBLE PRECISION WD,YD
      DIMENSION WD(15,7)
      SCALED FRACTION X,W,WT,Y,YTRUE,EABS
      INTEGER ALPHA
      LOGICAL SENSW
      COMMON /SHIFT/ISFT
C
      READ(6,12)NPAR,NREC,N,IWAIT,ISFT
   12 FORMAT(5I5)
      READ(6,16)(SCALE(I),I=1,16)
   16 FORMAT(8F10.4)
      READ(6,13) (IFM(I),I=1,40)
   13 FORMAT(40A2)
C
      CALL WEIGHT(NELEM,ALPHA,N1,N2,NW,W)
C
      IF(SENSW(2))WRITE(1,74)
      IF(.NOT.SENSW(2))WRITE(1,73)
   73 FORMAT(1H 16X,36HSINGLE PRECISION ELEMENT COMPUTATION///)
   74 FORMAT(1H 16X,36HDOUBLE PRECISION ELEMENT COMPUTATION///)
      DO 80 LOOP=1,NREC
      READ(6,IFM) (X(I),I=1,NPAR),YTRUE
      YSAVE(1)=YTRUE
      I=1
      DO 79 IS=1,4
      ISFT=IS-1
      DO 60 IN=1,6
      I=I+1
      NBITS=18-(2*IN)
      CALL ASSGN(NBITS)
      CALL YNET(X,NPAR,NELEM,ALPHA,N1,N2,NW,W,Y,NBITS)
      CALL DNET(X,NPAR,NELEM,ALPHA,N1,N2,NW,WD,YD,NBITS,ISFT,W)
      CALL STATS(IS,LOOP,IN,NELEM,NREC,W,WD)
   60 CONTINUE
   79 CONTINUE
   80 CONTINUE
      STOP
      END
```

PAGE 1 C STATS

```
      SUBROUTINE STATS(ISFT,LOOP,NBITS,NELEM,NREC,W,WD)
      DIMENSION AAE(6,4,4),MXAE(6,4,4),AAVT(6,4,4),AAV(6,4,4),MXAVT(6,4.
     14),MNAVT(6,4,4),MXAV(6,4,4),MNAV(6,4,4),IST(4),IFIN(4),
     2 W(15,8),WD(15,7),OUT(15,2),ERROR(15),SUM( 4)
```

```
      SCALED FRACTION W
      DOUBLE PRECISION WD
      REAL AAE,MXAE,AAVT,MXAVT,MNAVT,MXAV,MNAV
      LOGICAL SENSW
      COMMON /SAV/ OUT
      DATA IST(1),IFIN(1),IST(2),IFIN(2),IST(3),IFIN(3),IST(4),IFIN(4)
     1/1,8,9,12,13,14,15,15/
      DREC=FLOAT(NREC)
      SUM(1)=DREC*8.
      SUM(2)=DREC*4.
      SUM(3)=DREC*2.
      SUM(4)=DREC
C
C INITIALIZE ALL ARRAYS
C
      IF(ISFT.NE.1)GO TO 15
      IF(LOOP.NE.1)GO TO 15
      IF(NBITS.NE.1)GO TO 15
      DO 70 I=1,6
      DO 69 L=1,4
      DO 68 K=1,4
      AAE(I,L,K)=0.
      AAV(I,L,K)=0.
      MXAE(I,L,K)=0.
      AAVT(I,L,K)=0.
      MXAV(I,L,K)=0.
      MXAVT(I,L,K)=0.
   68 CONTINUE
   69 CONTINUE
   70 CONTINUE
   15 CONTINUE
C
C CONVERT SCALED FRACTION AND DOUBLE PRECISION ELEMENT OUTPUTSTO
C  COMMON MODE AND COMPUTE ABSOLUTE ERROR.
C
      DO 12 I=1,NELEM
      OUT(I,1)=W(I,7)
      OUT(I,1)=ABS(OUT(I,1))
      IF(NBITS.EQ.1)OUT(I,2)=WD(I,7)
      IF(SENSW(7))OUT(I,2)=WD(I,7)
      OUT(I,2)=ABS(OUT(I,2))
      E=OUT(I,1)-OUT(I,2)
      ERROR(I)=ABS(E)
   12 CONTINUE
C
C
      DO 25 L=1,4
      IA=IST(L)
      IB=IFIN(L)
      DO 20 I=IA,IB
      AAE(NBITS,L,ISFT)=AAE(NBITS,L,ISFT)+ERROR(I)
      IF(ERROR(I).GT.MXAE(NBITS,L,ISFT))MXAE(NBITS,L,ISFT)=ERROR(I)


PAGE  2     C     STATS

      AAVT(NBITS,L,ISFT)=AAVT(NBITS,L,ISFT)+OUT(I,2)
      AAV(NBITS,L,ISFT)=AAV(NBITS,L,ISFT)+OUT(I,1)
      IF(OUT(I,2).GT.MXAVT(NBITS,L,ISFT))MXAVT(NBITS,L,ISFT)=OUT(I,2)
      IF(OUT(I,1).GT.MXAV(NBITS,L,ISFT))MXAV(NBITS,L,ISFT)=OUT(I,1)
C
   20 CONTINUE
   25 CONTINUE
      IF(.NOT.SENSW(8))GO TO 191
      WRITE(1,189) AAE(NBITS,1,ISFT),MXAE(NBITS,1,ISFT)
```

```
  [illegible] FORMAT(TH 2F15.5)
  191 CONTINUE
C
C
      IF(LOOP.NE.NREC)GO TO 60
      IF(ISFT.NE.4) GO TO 60
      IF(NBITS.NE.6) GO TO 60
      DO 30 I=1,6
      DO 29 K=1,4
      DO 28 L=1,4
      AAE(I,L,K)=AAE(I,L,K)/SUM(L)
      IF(AAE(I,L,K).EQ.0.)GO TO 160
      AAE(I,L,K)=20.*ALOG10(AAE(I,L,K))
  160 CONTINUE
      IF(MXAE(I,L,K).EQ.0.)GO TO 161
      MXAE(I,L,K)=20.*ALOG10(MXAE(I,L,K))
  161 CONTINUE
      AAVT(I,L,K)=AAVT(I,L,K)/SUM(L)
      AAV(I,L,K)=AAV(I,L,K)/SUM(L)
   28 CONTINUE
   29 CONTINUE
   30 CONTINUE
C
C
      DO 81 ISF=1,4
      NSFT=ISF-1
      WRITE(1,59) NSFT
   59 FORMAT(///6HSHIFT ,I5/)
      DO 80 IL=1,4
      WRITE(1,49)
   49 FORMAT(//18HAVG ABS ERROR      /)
      WRITE(1,50)IL,(AAE(NB,IL,ISF),NB=1,6)
   50 FORMAT(I5,5X,6F10.5)
      WRITE(1,52)
   52 FORMAT(//13HMAX ABS ERROR /)
      WRITE(1,50)IL,(MXAE(NB,IL,ISF),NB=1,6)
      WRITE(1,51)
   51 FORMAT(//18HTRUE AVG ABS VALUE/)
      WRITE(1,50) IL,(AAVT(NB,IL,ISF),NB=1,6)
      WRITE(1,53)
   53 FORMAT(//13HAVG ABS VALUE /)
      WRITE(1,50) IL,(AAV(NB,IL,ISF),NB=1,6)
      WRITE(1,54)
   54 FORMAT(//18HTRUE MAX ABS VALUE /)
      WRITE(1,50) IL,(MXAVT(NB,IL,ISF),NB=1,6)
      WRITE(1,56)
   56 FORMAT(//13HMAX ABS VALUE /)
      WRITE(1,50) IL,(MXAV(NB,IL,ISF),NB=1,6)
   80 CONTINUE


PAGE  3     C     STATS

   81 CONTINUE
C
   60 RETURN
      END


PAGE  1     C     DNET


      SUBROUTINE DNET(X,NPAR,NELEM,ALPHA,NI,NO,NW,WD,YD,M,ISFT,W)
```

```
      DIMENSION X(1),ALPHA(1),I1(1),N2(1),NW(1),W(15,8),WTD(6),
     1  XD(50),WD(15,7)
      SCALED FRACTION X,XM,MASK,W,WM
      REAL XR,WR,RSFT,ROUND
      DOUBLE PRECISION XD,WD,X1D,X2D,WTD,DPELEM,DSFT,YD
      INTEGER ALPHA,LETTER
      DATA LETTER /1HX/
C
C MASK X'S AND W'S AND DEFINE DP X'S AND W'S.
C
      DO 20 I=1,NPAR
      XM=X(I)
      IF(M.EQ.16)GO TO 28
      XM=MASK(XM)
   28 XR=XM
      XD(I)=XR
   20 CONTINUE
C
      DO 26 N=1,6
      DO 25 IJK=1,NELEM
      WM=W(IJK,N)
      IF(M.EQ.16)GO TO 19
      WM=MASK(WM)
   19 WR=WM
   25 WD(IJK,N)=WR
   26 CONTINUE
C
C COMPUTE NETWORK
C
      DO 30 IJK=1,NELEM
      I=N1(IJK)
      J=N2(IJK)
      IF(ALPHA(IJK).NE.LETTER) GO TO 15
      X1D=XD(I)
      X2D=XD(J)
      GO TO 16
   15 X1D=WD(I,7)
      X2D=WD(J,7)
   16 N=NW(IJK)
      DO 18 NT=1,N
   18 WTD(NT)=WD(IJK,NT)
      WD(IJK,7)=DPELEM(X1D,X2D,N,WTD)
      RSFT=FLOAT(ISFT)
      IF(ISFT.EQ.0)GO TO 30
      RSFT=2.**RSFT
      DSFT=RSFT
      WD(IJK,7)=WD(IJK,7)*DSFT
   30 CONTINUE
      YD=WD(NELEM,7)
C
C ROUND ALL ELEMENT OUTPUTS TO M BITS.
C
      ROUND=2.**(-M-1)
      DO 40 IJK=1,NELEM
      WR=WD(IJK,7)


PAGE  2     C       DNET

      IF(M.EQ.16)GO TO 36
      WR=WR+ROUND
   36 CONTINUE
      IF(WR.GE..99990)WR=.99990
      IF(WR.LE.-.99990)WR=-.99990
      WM=WR
```

```
      DIMENSION(NW)
      W(IJK,8)=WM
   40 CONTINUE
C
      RETURN
      END
```

PAGE 1 C READ SCALED FRACTION WEIGHTS

```
      SUBROUTINE WEIGHT(NELEM,ALPHA,N1,N2,NW,W)
C
C THIS SUBPROGRAM READS IN A SCALED FRACTION NET
C
      DIMENSION ALPHA(1),W(15,7),N1(1),N2(1),NW(1)
      SCALED FRACTION W
      INTEGER ALPHA
      LOGICAL SENSW
C
      READ(6,11) NELEM
   11 FORMAT(I5)
      DO 30 IJK=1,NELEM
      READ(6,12)ALPHA(IJK),N1(IJK),N2(IJK),N,(W(IJK,I),I=1,N)
   12 FORMAT(A1,I2,1X,2I2,6S10)
      NW(IJK)=N
      IF(.NOT.SENSW(5)) GO TO 59
      WRITE(1,13)IJK,ALPHA(IJK),N1(IJK),N2(IJK),NW(IJK),(W(IJK,I),I=1,N
   13 FORMAT(1H I5,3X,A1,3I4,2X,6S7/)
   59 CONTINUE
   30 CONTINUE
      RETURN
      END
```

PAGE 1 C YNET

```
      SUBROUTINE YNET(X,NPAR,NELEM,ALPHA,N1,N2,NW,W,Y,M)
C
C YNET PERFORMS THE TRANSFORMATION Y=X * W
C
      DIMENSION X(1),ALPHA(1),N1(1),N2(1),NW(1),W(15,8),WT(6)
      INTEGER ALPHA,LETTER
      SCALED FRACTION X,W,WT,X1,X2,YELEM,Y,YELEM1
      LOGICAL SENSW
      DATA LETTER/1HX/
C
C
      DO 30 IJK=1,NELEM
      I=N1(IJK)
      J=N2(IJK)
      IF(ALPHA(IJK).NE.LETTER) GO TO 15
      X1=X(I)
      X2=X(J)
      GO TO 16
   15 X1=W(I,7)
      X2=W(J,7)
   16 N=NW(IJK)
      DO 18 NT=1,N
   18 WT(NT)=W(IJK,NT)
      W(IJK,7)=YELEM(X1,X2,N,WT)
```

```
      IF(SENSW(2))W(IJK,7)=YELEM(X1,X2,N,WT,M)
C
C SHIFT THE ELEMENT OUTPUT ---INTERMEDIATE SHIFT-----.C
C
      CALL SFT(W(IJK,7))
   30 CONTINUE
      Y=W(IJK,7)
      RETURN
      END
```

PAGE 1 C ASSIGN MASK

```
      SUBROUTINE ASSGN(N)
      COMMON MSK
      INTEGER M(16)
      DATA M(1),M(2),M(3),M(4),M(5),M(6),M(7),M(8),M(9),M(10),M(11),
     1 M(12),M(13),M(14),M(15),M(16)/-'00000,
     1    -'40000,-'20000,-'10000,
     2   -'04000,-'02000,-'01000,
     3   -'00400,-'00200,-'00100,
     4   -'00040,-'00020,-'00010,
     5   -'00004,-'00002,-'00001/
      MSK=M(N)
      RETURN
      END
```

PAGE 1 C MASK A SF TO DESIRED BITS

```
      FUNCTION MASK(X)
      SCALED FRACTION X
      SCALED FRACTION MASK
      INTEGER MSK
      COMMON MSK
C
C USE LOGICAL AND FUNCTION TO MASK N BITS OF WORD X
C
      LA  X
      AND MSK
      STA MASK
C
      RETURN
      END
```

PAGE 1 C YELEM

```
      FUNCTION YELEM(X1,X2,N,WT)
      DIMENSION WT(6)
      SCALED FRACTION X1,X2,WT,YELEM,TEMP,X1S,X2S,X12,T1,T2,T3,T4,T5,T6
      SCALED FRACTION MASK
      LOGICAL SENSW
      EXTERNAL MASK
C
C
C
```

```
C MASK ALL X'S AND W'S TO SPECIFIED BITS
C
      X1=MASK(X1)
      X2=MASK(X2)
      DO 20 K=1,N
      TEMP=WT(K)
   20 WT(K)=MASK(TEMP)
      IF(N.EQ.6)GO TO 22
      WT(5)=.00000S
      WT(6)=.00000S
   22 CONTINUE
C
C
C
C COMPUTE X SQUARES ,CROSS PROD,AND MASK.
C
      X1S=X1*X1
      X2S=X2*X2
        X12=X1*X2
      X1S=MASK(X1S)
        X2S=MASK(X2S)
      X12=MASK(X12)
C
C MULTIPLY W'S AND X'S AND MASK
C
      T1=WT(1)
      T2=WT(2)*X1
      T3=WT(3)*X2
      T4=WT(4)*X12
      T5=WT(5)*X1S
      T6=WT(6)*X2S
      T1=MASK(T1)
      T2=MASK(T2)
      T3=MASK(T3)
      T4=MASK(T4)
      T5=MASK(T5)
      T6=MASK(T6)
C
      YELEM=T1+T2+T3+T4+T5+T6
      RETURN
      END


PAGE  1     C   YELEM1


      FUNCTION YELEM1(X1,X2,N,WT,M)
      DIMENSION WT(6)
      SCALED FRACTION X1,X2,YELEM1,WT,SUMA,SUMS,T2A,T2Q,T3A,T3Q,
     1    T4A,T4Q,T5A,T5Q,T6A,T6Q,ROUND
      SCALED FRACTION MASK,TEMP
      LOGICAL SENSW
      EXTERNAL MASK
C
C MASK ALL W'S AND X'S.
C
      X1=MASK(X1)
      X2=MASK(X2)
      DO 20 K=1,N
      TEMP=WT(K)
   20 WT(K)=MASK(TEMP)
      IF(N.EQ.6)GO TO 22
      WT(5)=.0000S
      WT(6)=.0000S
```

```
   CONTINUE
C
      SUMA=WT(1)
      SUMQ=.0000S
C
C MULTIPLY X'S AND W'S TO FORM THE 6 TERMS. MULT2 SAVES THE 2T BIT
C PRODUCT IN THE LAST 2 ARGS. SAME WITH MULT3.
C
      CALL MULT2(WT(2),X1,T2A,T2Q)
      CALL MULT2(WT(3),X2,T3A,T3Q)
      CALL MULT3(WT(4),X1,X2,T4A,T4Q,M)
      CALL MULT3(WT(5),X1,X1,T5A,T5Q,M)
      CALL MULT3(WT(6),X2,X2,T6A,T6Q,M)
C
C
C
C ADD THE 6 DOUBLE PRECISION (2T) TERMS AND STORE DP RESULT IN SUMA,SUM
C
      CALL DPADD(SUMA,SUMQ,T2A,T2Q)
      CALL DPADD(SUMA,SUMQ,T3A,T3Q)
      CALL DPADD(SUMA,SUMQ,T4A,T4Q)
      CALL DPADD(SUMA,SUMQ,T5A,T5Q)
      CALL DPADD(SUMA,SUMQ,T6A,T6Q)
C
C ROUND TO T BIT POSITIONS AND MASK.
C
      CALL RND(SUMA,SUMQ,M)
      YELEMI=SUMA
      RETURN
      END


PAGE  1     C   C       DPELEM


      FUNCTION DPELEM(DX1,DX2,N,DWT)
      DIMENSION DWT(6)
      DOUBLE PRECISION DX1,DX2,DWT,DX1S,DX2S,DX12,DPELEM
C
C
C PERFORM ELEMENT COMPUTATION IN DOUBLE PRECISION.
C
      DX1S=DX1*DX1
      DX2S=DX2*DX2
      DX12=DX1* DX2
C
      DPELEM=DWT(1)+DWT(2)*DX1+DWT(3)*DX2+DWT(4)*DX12
     1   +DWT(5)*DX1S+DWT(6)*DX2S
      RETURN
      END


PAGE  1     C    MULT2


      SUBROUTINE MULT2(A,B,C1,C2)
      SCALED FRACTION A,B,C1,C2
C
C
      LA  A
      M   B
      STA C1
```

```
      OCT 026540
      STA C2
      RETURN
      END
```

PAGE 1 C MULT3

```
      SUBROUTINE MULT3(A,B,C,D1,D2,M)
      SCALED FRACTION A,B,C,D1,D2,ROUND
      SCALED FRACTION MASK
      LOGICAL SENSW
      EXTERNAL MASK
C
C M IS THE NUM OF COMPUTATIONAL BITS AVAILABLE.
C THE PROD OF 'A' AND 'B' ARE ROUNDED TO M BITS BEFORE MULTIPLYING C.
C
      RR=2.0**(-M-1)
      ROUND=RR
      IF(M.NE.16)GO TO 21
      D1=A*B
      GO TO 22
C
   21 CONTINUE
      LA A
      M B
      A ROUND
      STA D1
C
      D1=MASK(D1)
   22 CONTINUE
C
      LA D1
      M C
      STA D1
      OCT 026540
      STA D2
C
      RETURN
      END
```

PAGE 1 C DPADD

```
      SUBROUTINE DPADD(A1,A2,B1,B2)
      SCALED FRACTION A1,A2,B1,B2
C
C DPADD PERFORMS A DOUBLE PRECISION ADD OF A AND B , RESULTS STORED IN
C
C
      LA A2
      A B2
      OCT 026357
      A A1
      A B1
      STA A1
      OCT 026740
      OCT 026317
      STA A2
C
```

```
      RETURN
      END
```

PAGE 1 C

```
C
C   ROUND
      SUBROUTINE RND(A1,A2,N)
      SCALED FRACTION A1,ROUND ,MASK,A2
      EXTERNAL MASK
C
C RND WILL ROUND A1 TO N BITS AND MASK LEAST SIG BITS WITH ZEROS.
C
      IF(N.EQ.16)GO TO 20
      RR=2.0**(-N-1)
      ROUND=RR
      A1=A1+ROUND
      A1=MASK(A1)
      RETURN
C
   20 CONTINUE
      LA  A2
      OCT 026256
      A   A1
      STA A1
      RETURN
      END
```

PAGE 1 C SFT

```
      SUBROUTINE SFT(VALUE)
C SFT PERFORMS AN ARITHMETIC LEFT SHIFT ON ALL ELEMENT OUTPUTS.
C
      SCALED FRACTION VALUE
      REAL VAL,A
      COMMON /SHIFT/ISFT
C
      IF(ISFT.EQ.0)GO TO 10
C
C CHECK FOR OVERFLOW
C
      VAL=VALUE
      S=FLOAT(ISFT)
      A=VAL*2.** S
      IF(A.GE..9999)VALUE=.9999S
      IF(A.LE.-.9999)VALUE=-.9999S
      A=ABS(A)
      IF(A.GE..9999)GO TO 10
      GO TO  (1,2,3,4,5),ISFT
    1 CONTINUE
      LA  VALUE
      OCT 026001
      STA VALUE
      GO TO 10
    2 CONTINUE
      LA  VALUE
      OCT 026002
      STA VALUE
```

AD-A042 256 RCA GOVERNMENT COMMUNICATIONS SYSTEMS SOMERVILLE N J --ETC F/G 9/2
LSI ELECTRONICALLY PROGRAMMABLE ARRAYS: CONFIGURABLE POLYNOMIAL--ETC(U)
JUN 77 D HAMPEL, K PROST, R L BARRON F33615-73-C-1089
UNCLASSIFIED AFAL-TR-76-228 NL

4 OF 4
AD
A042256

END
DATE
FILMED
8-77

```
      GO TO 10
    3 CONTINUE
      LA   VALUE
      OCT 026003
      STA VALUE
      GO TO 10
    4 CONTINUE
        LA   VALUE
      OCT 026004
      STA VALUE
      GO TO 10
    5 CONTINUE
      LA   VALUE
      OCT 026005
      STA VALUE
   10 RETURN
      END

JOB CORRECT
```

APPENDIX J

An algorithm is derived below to estimate recursively the mean and the sum of the feature variances of patterns from a particular class.

The symbols have the following interpretation:

y - n-dimension pattern vector.

μ_N - centroid at the N^{th} iteration.

$\mu_{j,N}$ - estimate of the j^{th} element of μ at the N^{th} iteration.

A_N - sample covariance matrix of y at the N^{th} iteration.

$\sigma^2_{j,N}$ - sample variance of y_j at the N^{th} iteration.

The following is a procedure by which we can estimate recursively the mean and covariance matrix of y:

The sample covariance matrix of y at the N^{th} stage is,

$$(N+1)A_{N+1} \triangleq \begin{pmatrix} \sum_{i=1}^{N+1}\left(y_1(i)-\mu_1(i)\right)^2 & \sum_{i=1}^{N+1}\left((y_1-\mu_1)(y_2-\mu_2)\right)_i & \cdots & \sum_{i=1}^{N+1}\left((y_1-\mu_1)_i(y_n-\mu_n\right. \\ \sum_{i=1}^{N+1}\left((y_1-\mu_1)(y_2-\mu_2)\right)_i & \sum_{i=1}^{N+1}\left(y_2-\mu_2\right)_i^2 & \cdots & \sum_{i=1}^{N+1}\left((y_2-\mu_2)(y_n-\mu_n\right) \\ \vdots & \vdots & & \vdots \\ \sum_{i=1}^{N+1}\left((y_n-\mu_n)\ (y_1-\mu_1)\right)_i & \cdots & & \sum_{i=1}^{N+1}\left(y_n-\mu_n\right)_i^2 \end{pmatrix}$$

$$(N+1)\ \mathrm{Tr}\ A_{N+1} = \sum_{j=1}^{n}\sum_{i=1}^{N+1}(y_j-\mu_j)_i^2 = \sum_{j=1}^{n}\sigma_{j,N+1}^2\cdot(N+1) \qquad (1a)$$

$$N\ \mathrm{Tr}\ A_N = \sum_{j=1}^{n}\sum_{i=1}^{N}(y_j-\mu_j)_i^2 = \sum_{j=1}^{N}\sigma_{j,N}^2\cdot N \qquad (1b)$$

$$\mathrm{Tr}\ A_{N+1} = \sum_{j=1}^{n}\sigma_{j,N+1}^2 \qquad (2a)$$

$$\mathrm{Tr}\ A_N = \sum_{j=1}^{n}\sigma_{j,N}^2 \qquad (2b)$$

Now

$$\sum_{i=1}^{N+1}y_j^2(i) = (N+1)\left(\sigma_{j,N+1}^2 + \mu_{j,N+1}^2\right) \qquad (3)$$

and

$$\sum_{i=1}^{N} y_j^2(i) = N\left(\sigma_{j,N}^2 + \mu_{j,N}^2\right) \tag{4}$$

Subtracting (4) and (3)

$$y_{j,N+1}^2 = (N+1)\left[\sigma_{jN+1}^2 + \mu_{j,N+1}^2\right] - N\left[\sigma_{j,N}^2 + \mu_{j,N}^2\right] \tag{5}$$

dropping the subscript j

$$y_{N+1}^2 = (N+1)\left(\sigma_{N+1}^2 + \mu_{N+1}^2\right) - N\left(\sigma_N^2 + \mu_N^2\right) \tag{6}$$

$$\sigma_{N+1}^2 = \left(\frac{N}{N+1}\right)\left(\sigma_N^2 + \mu_N^2\right) + \frac{y_{N+1}^2}{N+1} - \mu_{N+1}^2 \tag{7}$$

But

$$\mu_{N+1} = \left(\frac{N}{N+1}\right)\mu_N + \frac{y_{N+1}}{N+1} \tag{8}$$

$$\mu_{N+1}^2 = \left(\frac{N}{N+1}\right)^2 \cdot \mu_N^2 + \frac{y_{N+1}^2}{(N+1)^2} + \frac{2N}{(N+1)^2}\,\mu_N\, y_{N+1} \tag{9}$$

Substituting in (7)

$$\sigma_{N+1}^2 = \left(\frac{N}{N+1}\right)\left(\sigma_N^2 + \mu_N^2\right) + \frac{y_{N+1}^2}{N+1} - \left(\frac{N}{N+1}\right)^2 \mu_N^2 - \frac{y_{N+1}^2}{(N+1)^2} - \frac{2N}{(N+1)^2}\,\mu_N\, y_{N+1} \tag{10}$$

$$\sigma^2_{N+1} = \left(\frac{N}{N+1}\right) \sigma^2_N + \frac{\mu^2_N}{(N+1)^2}\left[\frac{N}{N+1} - \frac{(N)^2}{(N+1)^2}\right]$$

$$+\frac{y^2_{N+1}}{N+1}\left(\frac{N}{N+1}\right) - \frac{2N}{(N+1)^2}\,\mu_N\, y_{N+1} \qquad (11)$$

$$= \left(\frac{N}{N+1}\right) \sigma^2_N + \frac{N}{(N+1)^2}\,(y_{N+1} - \mu_N)^2 \qquad (12)$$

$$(N+1)\,\sigma^2_{N+1} = N\sigma^2_N + \left(\frac{N}{N+1}\right)(y_{N+1} - \mu_N)^2 \qquad (13)$$

The (N+1)st estimate of σ^2_j is related to the N^{th} estimate of σ^2_j through the above relationship. Thus:

$$\sigma^2_{j,N+1} = \left(\frac{N}{N+1}\right) \sigma^2_{j,N} + \frac{N}{(N+1)^2}\,(y_{N+1} - \mu_{j,N})^2 \qquad (14)$$

$$\mathrm{Tr}\,(A_{N+1}) = \sum_{j=1}^{n} \sigma^2_{j,N+1} = \left(\frac{N}{N+1}\right)\left[\sum_{j=1}^{n} \sigma^2_{j,N} + \frac{1}{N+1}\sum_{j=1}^{n} (y_{j,N+1} - \mu_{j,N})^2\right] \qquad (15)$$

This can be set up on the computer with an initial

$$\sum_{j=1}^{n} \sigma_{j,0}^2 = 1 \quad \text{or } 0$$

The ratio of the traces of the sampled feature variance matrices of the two classes can be estimated recursively:

$$Tr(A_{N+1}) = \left(\frac{N}{N+1}\right) \cdot \left[Tr(A_N) + \frac{1}{N+1} \sum_{j=1}^{n} (y_{j,N+1} - \mu_{j,N})^2 \right] \tag{16}$$

and:

$$\mu_{j,N+1} = \left(\frac{N}{N+1}\right) \cdot \mu_{j,N} + \frac{y_{j,N+1}}{N+1} \tag{17}$$

$$y_N \triangleq \left\{ y_{j,N} \right\} ; \qquad \mu_N \triangleq \left\{ \mu_{j,N} \right\} \quad ; \; j = 1, \ldots, n$$

$$Tr(A_{N+1}) = \left(\frac{N}{N+1}\right) Tr(A_N) + \left(\frac{1}{N+1}\right) (y_{N+1} - \mu_N)' (y_{N+1} - \mu_N) \tag{18}$$

and

$$\mu_{N+1} = (N/N+1)\, \mu_N + \frac{y_{N+1}}{N+1} \tag{19}$$

APPENDIX K

Software for CPA Element Brassboard

Listing of Driver, D.66

Service Routine, POLY

Applications Program

APPLICATION PROGRAM

```
      DIMENSION WP3(9,6),WP2(6,4),WBUF(8),RBUF(15),DBUF(4),DATA(10)
      DIMENSION DATIN(10)
1     WRITE(1,5)
5     FORMAT("PUT WEIGHT TAPE IN READER AND PRESS RUN")
      PAUSE
      READ(5,*)((WP3(I,J),J=1,6),I=1,9)
      READ(5,*)((WP2(I,J),J=1,4),I=1,6)
9     WRITE(1,10)
10    FORMAT("PUT DATA TAPE IN READER AND PRESS RUN")
      PAUSE
      READ(5,*)(DATIN(M),M=1,10)
      DATA(1)=DATIN(1)
      DATA(2)=DATIN(2)
      DATA(3)=DATIN(5)
     DATA(4)=DATIN(10)
      DATA(5)=DATIN(7)
      DATA(6)=DATIN(10)
      DATA(7)=DATIN(6)
      DATA(8)=DATIN(10)
      DATA(9)=DATIN(5)
      DATA(10)=DATIN(6)
      DO 31 J=1,5
      DO 21 I=1,6
      IV=7-I
21    WBUF(IV)=WP3(J,I)
      DBUF(2)=DATA(2*J-1)
      DBUF(1)=DATA(2*J)
31    CALL POLY(3,DBUF(1),WBUF(1),RBUF(J))
     DATA(1)=RBUF(1)
      DATA(2)=RBUF(4)
     DATA(3)=RBUF(1)
      DATA(4)=RBUF(5)
      DATA(5)=RBUF(3)
      DATA(6)=RBUF(5)
      DATA(7)=RBUF(2)
      DATA(8)=RBUF(3)
      DO 32 J=6,9
      DO 22 I=1,6
      IV=7-I
22    WBUF(IV)=WP3(J,I)
      DBUF(2)=DATA(2*J-11)
     DBUF(1)=DATA(2*J-10)
32    CALL POLY(3,DBUF(1),WBUF(1),RBUF(J))
     DO 23 I=1,4
      IV=5-I
23    WBUF(IV)=WP2(1,I)
      DO 231 I=1,4
      IV=9-I
231   WBUF(IV)=WP2(2,I)
      DBUF(2)=RBUF(7)
      DBUF(1)=RBUF(9)
```

```
      DBUF(4)=RBUF(7)
       DBUF(3)=RBUF(8)
       CALL POLY(2,DBUF(1),WBUF(1),RBUF(10))
      DO 232 I=1,4
      IV=5-I
232    WBUF(IV)=WP2(3,I)
       DBUF(2)=RBUF(6)
       DBUF(1)=RBUF(7)
       DBUF(4)=0.0
       DBUF(3)=0.0
       CALL POLY(2,DBUF(1),WBUF(1),RBUF(12))
       DO 24 I=1,4
       IV=5-I
24     WBUF(IV)=WP2(4,I)
       DO 241 I=1,4
      IV=9-I
241    WBUF(IV)=WP2(5,I)
       DBUF(2)=RBUF(10)
       DBUF(1)=RBUF(12)
       DBUF(4)=RBUF(11)
       DBUF(3)=RBUF(12)
       CALL POLY(2,DBUF(1),WBUF(1),RBUF(13))
       DO 25 I=1,4
       IV=5-I
25     WBUF(IV)=WP2(6,I)
       DBUF(2)=RBUF(13)
       DBUF(1)=RBUF(14)
       DBUF(4)=0.0
       DBUF(3)=0.0
       CALL POLY(2,DBUF(1),WBUF(1),RBUF(15))
       WRITE(1,30)RBUF(15)
30     FORMAT("RESULT = ",E13.6)
       GO TO 9
       STOP
       END
       END$
```

```
0001  00000                    NAM POLYS
0002                           ENT POLY
0003                           EXT .IOC.,.ENTR,.STOP,ENDIO
0004  00000 000000  ARGS       BSS 4
0005  00004 000000  POLY       NOP
0006  00005 016002X            JSB .ENTR
0007  00006 000000R            DEF ARGS
0008  00007 166000R            LDB ARGS,I
0009  00010 016043R            JSB DWA
0010  00011 032145R            IOR SGN
0011  00012 072152R            STA D+4
0012  00013 062144R            LDA ARGSA
0013  00014 066143R            LDB DESTA
0014  00015 105777            MVW TWO
      00016 000162R
      00017 000000
0015  00020 062003R            LDA ARGS+3
0016  00021 072154R            STA D+6
0017  00022 016001X            JSB .IOC.
0018  00023 010011            OCT 10011
0019  00024 016115R            JSB ERRX
0020  00025 000146R            DEF D
0021  00026 B77773            OCT -5
0022  00027 072155R            STA BSCT
0023  00030 016004X            JSB ENDIO
0024  00031 000032R            DEF *+1
0025  00032 016001X READ       JSB .IOC.
0026  00033 020011            OCT 20011
0027  00034 016115R            JSB ERRX
0028  00035 000153R            DEF D+5
0029  00036 177776            OCT -2
0030  00037 072155R            STA BSCT
0031  00040 016004X            JSB ENDIO
0032  00041 000042R            DEF *+1
0033  00042 126004R            JMP POLY,I
0034  00043 000000  DWA        NOP
0035  00044 056162R            CPB TWO
0036  00045 026055R            JMP TO
0037  00046 056163R            CPB THREE
0038  00047 026066R            JMP TRE
0039  00050 056164R            CPB FOUR
0040  00051 026077R            JMP FOR
0041  00052 056165R            CPB SIX
0042  00053 026106R            JMP SX
0043  00054 026134R            JMP ERRX1
0044  00055 076153R TO         STB AA
0045  00056 066162R            LDB TWO
0046  00057 076153R            STB AA
0047  00060 005000            BLS
0048  00061 076146R            STB D
0049  00062 005000            BLS
0050  00063 076147R            STB W
0051  00064 062157R            LDA P2
0052  00065 126043R            JMP DWA,I
0053  00066 006404  TRE        CLB,INB
0054  00067 076153R            STB AA
0055  00070 005000            BLS
```

```
0056  00071 076146R        STB D
0057  00072 005000         BLS
0058  00073 046162R        ADB TWO
0059  00074 076147R        STB W
0060  00075 062160R        LDA P3
0061  00076 126043R        JMP DWA,I
0062  00077 005100  FOR    BRS
0063  00100 076147R        STB W
0064  00101 005100         BRS
0065  00102 076146R        STB D
0066  00103 002400         CLA
0067  00104 076153R        STB AA
0068  00105 126043R        JMP DWA,I
0069  00106 066162R SX     LDB TWO
0070  00107 076147R        STB W
0071  00110 005000         BLS
0072  00111 076146R        STB D
0073  00112 076153R        STB AA
0074  00113 062161R        LDA P6
0075  00114 126043R        JMP DWA,I
0076  00115 000000  ERRX   NOP
0077  00116 006021         SSB,RSS
0078  00117 026134R        JMP ERRX1
0079  00120 062155R        LDA BSCT
0080  00121 052164R        CPA FOUR
0081  00122 026125R        JMP *+3
0082  00123 036155R        ISZ BSCT
0083  00124 062115R        LDA ERRX
0084  00125 042156R        ADA M3
0085  00126 124000         JMP A,I
0086  00127 002400         CLA
0087  00130 072155R        STA BSCT
0088  00131 062140R        LDA I
0089  00132 066141R        LDB BS
0090  00133 016003X        JSB .STOP
0091  00134 062140R ERRX1  LDA I
0092  00135 066142R        LDB UN
0093  00136 026132R        JMP ERRX1-2
0094  00137 077777  MPOS   OCT 77777
0095  00140 000012  I      OCT 12
0096  00141 041123  BS     ASC 1,BS
0097  00142 052516  UN     ASC 1,UN
0098  00143 000150R DESTA  DEF D+2
0099  00144 000001R ARGSA  DEF ARGS+1
0100  00145 100000  SGN    OCT 100000
0101  00146 000000  D      BSS 7
0102  00155 000000  BSCT   NOP
0103  00156 177775  M3     OCT -3
0104  00157 030000  P2     OCT 30000
0105  00160 020000  P3     OCT 20000
0106  00161 010000  P6     OCT 10000
0107  00162 000002  TWO    OCT 2
0108  00163 000003  THREE  OCT 3
0109  00164 000004  FOUR   OCT 4
0110  00165 000006  SIX    OCT 6
0111  00000         A      EQU 0
0112  00001         B      EQU 1
```

```
0113  00153          AA      EQU D+5
0114  00147          W       EQU D+1
0115  00166 000000   CTR     NOP
0116  00167 000170R ANSAD   DEF *+1
0117  00170 000000   ANSBF   BSS 8
0118  00200 000000   TEMP    NOP
0119  00201 000202R INTAD   DEF *+1
0120  00202 000000   INTBF   BSS 24
0121  00232 000000   TEM1    NOP
0122                         END
**  NO ERRORS*
```

```
0001  00000                NAM D.66
0002                       ENT D.66,I.66
0003*  THIS BCS DRIVER IS DESIGNED TO INITIATE ,CONTINUE AND COMPLETE
0004*  BOTH READ AND WRITE TRANSFERS TO & FROM THE TCS-039 BASED
0005*  PROGRAMMABLE ARRAY MODULE
0006*
0007*
0008*  SAMPLE CALL
0009*        WRITE                              READ
0010*      JSB .IOC                           JSB .IOC
0011*      OCT ****                            OCT ****
0012*      JMP ERR                            JMP ERR
0013*      DEF D                              DEF D+5
0014*      OCT -5                             OCT -2
0015*
0016*D     +# OF DATA WORDS
0017*D+1   +# OF WEIGHTS
0018*D+2   ADDRESS OF 1ST DATA WORD
0019*D+3    ADDRESS OF 1ST WEIGHT
0020*D+4   POLYNOMIAL TYPE
0021*D+5   +# OF ANSWER WORDS
0022*D+6   ADDRESS OF 1ST ANSWER WORD DESTINATION
```

```
0025*
0026*          INITIATOR SECTION
0027*
0028  00000 000000  D.66  NOP       ENTRY &EXIT
0029  00001 072242R        STA SAVA       SAVE EQT ENTRY ADDRESS
0030  00002 076243R        STB SAVB        SAVE WD 2 ADDRESS
0031  00003 160001         LDA B,I         GET WD 2 OF RRQUEST
0032  00004 001700         ALF             ROTATE
0033  00005 012274R        AND M17           AND ISOLATE RCODE
0034  00006 002002         SZA              =0??
0035  00007 026015R        JMP D.XX        NO, THEN CONTINUE
0036*
0037*          RCODE=0 ---TERMINATE OPERATION
0038*
0039  00010 126000R I.1    JMP D.66,I      THIS IS CLC AFTER 1ST OPERATION
0040  00011 072242R        STA SAVA
0041  00012 062000R        LDA D.66        SET EXIT OF CONTINUATOR
0042  00013 072152R        STA I.66           SECTION TO .IOC
0043  00014 026222R        JMP STAT        NOW GO TO CONTINUATOR AND CLEAR
0044*
0045*          DRIVER BUSY TEST
0046*
0047  00015 066253R D.XX   LDB DFLG        IF DRIVER BUSY
0048  00016 006002         SZB              (DFLG NOT=0), THEN
0049  00017 026077R        JMP REJB           REJECT REQUEST
0050T
0051*          TEST FOR ILLEGAL REQUEST CODES
0052*
0053  00020 052270R        CPA B1       =1 THEN
0054  00021 026025R        JMP WRITE          WRITE REQUEST
0055  00022 052271R        CPA B2       =2 THEN
0056  00023 026060R        JMP READ           READ REQUEST
0057  00024 026076R        JMP REJC        IF NEITHER THEN REJECT REQUEST
```

```
0059*
0060*        WRITE REQUEST PROCESSING
0061*
0062  00025 016102R WRITE JSB SETIO      GO AND CONFIGURE I/O INSTS
0063  00026 002404        CLA,INA       SET READ/WRITE
0064  00027 072254R       STA R/W          FLAG NOT=0
0065  00030 016133R       JSB GETBF     GO GET BUFFER START (LENGTH =5)
0066  00031 162251R       LDA BUF,I     GET # OF DATA WORDS
0067  00032 001000        ALS        MPY BY 2 TO GET # OF WDS TO TRANSFER
0068  00033 003004        CMA,INA       MAKE NEGATIVE AND SAVE
0069  00034 072255R       STA DCNT      AS DATA COUNTER
0070  00035 036251R       ISZ BUF       INDEX TO NEXT WORD
0071  00036 B62251R       LDA BUF,I     GET # OF WEIGHTS
0072  00037 001000        ALS        X2 TO GET WDS
0073  00040 003004        CMA,INA       MAKE NEGATIVE AND
0074  00041 072261R       STA WCNT      SAVE AS WEIGHT COUNTER
0075  00042 036251R       ISZ BUF       INDEX TO NEXT WORD
0076  00043 162251R       LDA BUF,I     GET ADDRESS OF 1ST DATA WORD
0077  00044 072256R       STA DADD       AND SAVE
0078  00045 036251R       ISZ BUF       INDEX TO NEXT WORD
0079  00046 162251R       LDA BUF,I      GET ADDRESS OF 1ST WEIGHT
0080  00047 072262R       STA WADD       AND SAVE
0081  00050 036251R       ISZ BUF       INDEX ONCE MORE
0082  00051 162251R       LDA BUF,I     GET POLY TYPE
0083  00052 072250R       STA TYPE        AND SAVE
0084  00053 102600  I.3   OTA 0         OUTPUT COMMAND WORD
0085  00054 072253R       STA DFLG      SET TO INDICATE BUSY IE. NOT=0
0086  00055 002400        CLA           SET FOR OPERATION INITIATED
0087  00056 103700  I.4   STC 0,C       START DEVICE
0088  00057 126000R       JMP D.66,I    -EXIT- BACK TO .IOC
```

```
0090*
0091*          READ REQUEST PROCESSING
0092*
0093  00060 016102R READ  JSB SETIO       GO AND CONFIGURE I/O INSTS
0094  00061 002400        CLA             SET R/W FLAG
0095  00062 072254R       STA R/W             =0 TO INDICATE READ
0096  00063 016133R       JSB GETBF       GO GET BUFFER START (LENGTH =2)
0097  00064 162251R       LDA BUF,I       GET # OF ANSWERS
0098  00065 001000        ALS         X2 TO GET WDS
0099  00066 003004        CMA,INA         MAKE NEGATIVE
0100  00067 072257R       STA ANSCT        AND SAVE AS COUNTER
0101  00070 036251R       ISZ BUF          INDEX TO NEXT WORD
0102  00071 162251R       LDA BUF,I       GET ADDRESS OF 1ST ANSWER DESTI
0103  00072 072260R       STA ANSAD         AND SAVE
0104  00073 062250R       LDA TYPE        GET COMMAND WORD
0105  00074 032272R       IOR .14.        ADD READ FLAG
0106  00075 026053R       JMP I.3         GO OUTPUT IT AND RETURN TO .IOC
```

```
0108*
0109*          REJECT SECTION
0110*
0111  00076 006401  REJC  CLB,RSS          RCODE ERROR
0112  00077 066273R REJB  LDB M15           DRIVER/DEVICE BUSY =(B)SIGN =1
0113  00100 002404        CLA,INA           SET A NOT=0
0114  00101 126000R       JMP D.66,I        =EXIT- TO .IOC
0115*
0116*          I/O CONFIGURATION ROUTINE
0117*
0118  00102 000000  SETIO NOP               ENTRY &EXIT
0119  00103 162242R       LDA SAVA,I        GET WORD 1 OF EQT ENTRY
0120  00104 012275R       AND M77           ISOLATE SELECT CODE
0121  00105 032263R       IOR SFSI          COMBINE WITH SFS INSTRUCTION
0122  00106 072121R       STA I.2           SAVE
0123  00107 022267R       XOR LIAM          MAKE LIA INST
0124  00110 072164R       STA I.6           SAVE
0125  00111 022265R       XOR CLCM          MAKE CLC INST
0126  00112 072010R       STA I.1           SAVE
0127  00113 022266R       XOR STCM          MAKE STC X,C INST
0128  00114 072056R       STA I.4
0129  00115 072211R       STA I.8           SAVE
0130  00116 022264R       XOR OTAM          NOW MAKE OTA INST
0131  00117 072053R       STA I.3           SAVE
0132  00120 072202R       STA I.7           SAVE AGAIN
0133  00121 102300  I.2    SFS 0            IF FLAG NOT SET THEN
0134  00122 026077R       JMP REJB              REJECT REQUEST
0135*
0136*          SET EQT BUSY FLAG
0137*
0138  00123 036242R       ISZ SAVA          SET ADDRESS TO WD 2 OF EQT
0139  00124 162242R       LDA SAVA,I        ENTRY, SET BIT 15 OF
0140  00125 032273R       IOR M15           WD 2=1 (AFIELD=2) TO
0141  00126 172242R       STA SAVA,I        SAY BUSY
0142  00127 062242R       LDA SAVA          SETA ADDRESS OF
0143*
0144*          STORE ADDRESS OF EQT WORD 3 IN DRIVER
0145*
0146  00130 002004        INA               EQT WORD 3
0147  00131 072247R       STA EQTA          IN EQTA
0148  00132 126102R       JMP SETIO,I       AND BACK TO CALLER
```

```
0150*
0151*          ROUTINE TO GET USERS BUFFER ADDRESS AND SAVE IT
0152*
0153  00133 000000  GETBF NOP                ENTRY & EXIT
0154  00134 036243R        ISZ SAVB          INDEX TO
0155  00135 036243R        ISZ SAVB         WORD 4 OF REQUEST
0156  00136 062243R        LDA SAVB          GET WORD 4
0157  00137 160000         LDA A,I              OF REQUEST
0158  00140 001275         RAL,CLE,SLA,ERA    (IF INDIRECT GET
0159  00141 026137R        JMP *-2             EFFECTIVE ADDRESS)
0160  00142 072251R        STA BUF              SAVE
0161*
0162*        GET BUFFER LENGTH
0163*
0164  00143 036243R        ISZ SAVB          INDEX TO WORD 5 OF REQUEST
0165  00144 162243R        LDA SAVB,I         GET WORD 5
0166  00145 003004         CMA,INA            MAKE POSITIVE
0167  00146 006400         CLB
0168  00147 176247R        STB EQTA,I         CLEAR OUT XMISSION LOG
0169  00150 072252R        STA LENG          SAVE BUFFER LENGTH
0170  00151 126133R        JMP GETBF,I       -EXIT- TO CALLER
```

```
0172*
0173*      I.66  CONTINUATOR SECTION
0174*
0175*
0176*
0177  00152 000000  I.66   NOP            ENTRY & EXIT
0178  00153 072244R        STA SAVAX        SAVE A
0179  00154 076245R        STB SAVBX             B
0180  00155 001520         ERA,ALS               E
0181  00156 102201         SOC                   &
0182  00157 002004         INA
0183  00160 072246R        STA SAVEX           OVERFLOW
0184*
0185*
0186  00161 062254R        LDA R/W          IS THIS A READ
0187  00162 000010         SLA               OR WRITE??
0188  00163 026174R        JMP .WR             WRITE
0189*
0190*       READ PROCESSING
0191*
0192  00164 102500  I.6    LIA 0            GET ANSWER FROM DEVICE
0193  00165 003000         CMA              INVERT DATA FOR KAL
0194  00166 172260R        STA ANSAD,I      DELIVER TO SERVICE ROUTINE
0195  00167 036257R        ISZ ANSCT       INDEX TO NEXT
0196  00170 002001         RSS
0197  00171 026222R        JMP STAT          DONE FINISH UP
0198  00172 036260R        ISZ ANSAD        BUOP ADDRESS TO NEXT WORD
0199  00173 026203R        JMP XIT          GO TO EXIT ROUTINE
0200*
0201*       WRITE PROCESSING
0202*
0203  00174 062255R .WR    LDA DCNT         GET DATA COUNTER
0204  00175 002003         SZA,RSS            =0??
0205  00176 026213R        JMP .WRI          YES GOTO WEIGHT ROUTINE
0206  00177 162256R        LDA DADD,I        NO, GET ANOTHER DATA POINT
0207  00200 036255R        ISZ DCNT          MORE DATA??
0208  00201 036256R        ISZ DADD          YES, SET FOR NEXT POINT
0209  00202 102600  I.7    OTA 0              OUTPUT WORD
0210*
0211*       REGISTER RESTORE SECTION
0212*
0213  00203 062246R XIT    LDA SAVEX          RESTORE
0214  00204 103101         CLO                E
0215  00205 000036         SLA,ELA            OVERFLOW
0216  00206 102101         STF 1              A
0217  00207 062244R        LDA SAVAX           AND B AT
0218  00210 066245R        LDB SAVBX             TIME OF INTERRUPT
```

```
0220*
0221*       EXIT SECTION
0222*
0223  00211 103700  I.8    STC 0,C          START DEVICE FOR NEXT WORD (OR
0224  00212 126152R        JMP I.66,I       - EXIT -
0225  00213 062261R .WRI   LDA WCNT          GET WEIGHT COUNTER
0226  00214 002003         SZA,RSS           =0??
0227  00215 026222R        JMP STAT          YES FININ
0228  00216 162262R        LDA WADD,I        NO! GET NEXT WEIGHT
0229  00217 036261R        ISZ WCNT          MORE??
0230  00220 036262R        ISZ WADD          YES SET FOR NEXT TIME
0231  00221 026202R        JMP I.7           NO GO TO OUTPUT ROUTINE
```

```
0233*
0234*        STATUS SECTION
0235*
0236*        UPDATE XMISSION LOG
0237*
0238  00222 062254R STAT  LDA R/W         GET READ WRITE FLAG
0239  00223 002002        SZA             READ OR WRITE
0240  00224 042271R       ADA B2            ADD 2
0241  00225 042271R       ADA B2             ADD 2 AGAIN IF WRITE
0242  00226 172247R       STA EQTA,I       RESTORE WORD 3 OF EQT
0243*
0244*           UPDATE STATUS SECTION
0245*
0246  00227 003400        CCA             SET ADDRESS FOR WORD
0247  00230 042247R       ADA EQTA           2 OF
0248  00231 072251R       STA BUF             EQT
0249  00232 162251R       LDA BUF,I          GET WORD 2
0250  00233 012276R       AND MST              REMOVE PREVIOUS STATUS
0251  00234 172251R       STA BUF,I          RESTORE WORG 2 OF EQT
0252*
0253*            CLEAR DRIVER BUSY FLAG & EXIT
0254*
0255  00235 002400        CLA                CLEAR
0256  00236 072253R       STA DFLG                DRIVER BUSY FLAG
0257  00237 062010R       LDA I.1            SET CLC
0258  00240 072211R       STA I.8              IN EXIT SECTION
0259  00241 026203R       JMP XIT              ==DONE==
```

```
0261*
0262*
0263*          CONSTANT & STORAGE SECTION
0264   00000            A      EQU 0
0265   00001            B      EQU 1
0266   00242 000000     SAVA   NOP
0267   00243 000000     SAVB   NOP
0268   00244 000000     SAVAX  NOP
0269   00245 000000     SAVBX  NOP
0270   00246 000000     SAVEX  NOP
0271*
0272   00247 000000     EQTA   NOP
0273   00250 000000     TYPE   NOP
0274   00251 000000     BUF    NOP
0275   00252 000000     LENG   NOP
0276   00253 000000     DFLG   NOP
0277   00254 000000     R/W    NOP
0278   00255 000000     DCNT   NOP
0279   00256 000000     DADD   NOP
0280   00257 000000     ANSCT  NOP
0281   00260 000000     ANSAD  NOP
0282   00261 000000     WCNT   NOP
0283   00262 000000     WADD   NOP
0284*
0285   00263 102300     SFSI   SFS 0
0286   00264 001100     OTAM   OCT 1100
0287   00265 004200     CLCM   OCT 4200
0288   00266 005000     STCM   OCT 5000
0289   00267 000600     LIAM   OCT 600
0290*
0291   00270 000001     B1     OCT 1
0292   00271 000002     B2     OCT 2
0293   00272 040000     .14.   OCT 40000
0294   00273 100000     M15    OCT 100000
0295   00274 000017     M17    OCT 17
0296   00275 000077     M77    OCT 77
0297   00276 037400     MST    OCT 37400
0298*
0299                     END
** NO ERRORS*
```

REFERENCES

1. Basic and Applied Research in Industrial Process Adaptive Controls: First Quarterly Progress Report for Armco Steel Corporation (First Year), Adaptronics, Inc. Report under Purchase Order Res. 45, 152, July 1, 1970.

2. Widrow, B., Gupta, N. K. and Maitra, S., "Punishment/Reward: Learning with a Critic in Adaptive Threshold Systems," IEEE Trans. on Systems, Man and Cyberhetics, Vol. SMC-3, No. 5 Sept. 1973, pp 445 - 465.

3. Hoff, M. E. Jr., Learning Phenomena in Networks of Adaptive Switching Circuits, Tech. Rep. No. 1554-1 July 1962 Stanford Electronics Lab.

4. Jones, E. R. "A Progressive Learning Classifier" Tech. Rep. No. 6783-1, Dec. 1966, Stanford Electronics Lab.

5. Amari, Shun-Ichi, "Learning Patterns and Patterns Sequences by Self-Organizing Nets of Treshold Elements," IEEE Transactions on Computer, Vol. C-21, No. 11, Nov. 1972, pp 1197-1206.

6. Golay, M. J. E., Hexagonal Parallel Pattern Transformations, IEEE Transactions on Computer Vol. C-18 No. 8 Aug. 1969 pp 733-740.

7. Kautz, W. H., "Cellular Logic-in-Memory Arrays," IEEE Transactions on Computers, Vol. C-18, No. 8, Aug. 1969, pp 719-727.

8. Adaptronics Inc. Third Quarterly Progress Report for Armco Steel Corp. (Second Year), Jan. 1, 1972.

9. Cleveland, D., "Hardware Realization of Trainable Multivariable Nonlinar Transformations", presented to Computer Designer Conference, Anaheim, Calif. January 19-21, 1971.

10. Joseph, E. C., "Innovations in heterogeneous and homegeneous distributed-function architectures," Computer, Vol. 7, No. 3, Mar. 1973, pp 17-24.

11. Hampel, D., Blasco, R. W., and Barron, R. L., "LSI Electronically Programmable Arrays," Interim Report 1089-1, April 1974.

12. Decaire, J. A., "Desription/Specifications, High Speed Micro-Signal-Processor Study," RFP F33615-76-1249, AFAL, Dec. 16, 1975.

13. Kilpatrick, P. S., W. C. Marshall, and E. A. Scales, All Semiconductor Distributed Aerospace Processor/Memory Study, Vol. I: Avionics Processing Requirements, AFAL-TR-73-226, Vol. I, Aug. 1973.

14. Brigham, E. O., The Fast Fourier Transform, Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1974, pp. 202-206.

15. Hampel, D., R. W. Blasco, and R. L. Barron, LSI Electronically Programmable Arrays, Interim Report 1089-1, RCA and Adaptronics, Inc., for AFAL/TEA, Apr. 1974.

16. Piekarski, J., and J. R. Reid, Design of a CMOS, Fast Fourier Transform Module for Digital Signal Processing Applications, Naval Surface Weapons Center, NSWC/WOL/TR 75-55, Apr. 18, 1975.

17. Ivakhnenko, A. G., "Polynomial Theory of Complex Systems," IEEE Transactions on Systems, Man and Cybernetics, Vol. SMC-1, 4 Oct. 1971, pp 364-378.

18. Hampel, D., McGuire, K. E., Prost, K. J., "CMOS/SOS Serial-Parallel Multiplier," IEEE Journal of Solid-State Circuits, Vol. SC-10, No. 5, Oct. 1975, pp 307-314.

19. Baugh, C. R., Wooley, B. A., "Digital Multiplier Power Estimates," IEEE Transaction on Communications, Vol. COM-23, No. 11, Nov. 1975, pp 1287-1288.

20. Lyon, R. F., "Two's Complement Pipeline Multipliers," IEEE Transaction on Communications, Vol. COM-24, No. 4, April 1976, pp 418-425.

21. Mick, J. R., Springer, J., "Single-Chip Multiplier Expands Digital Role in Signal Processing," Electronics, May 13, 1976, pp 103-108.

22. Kaiser, H. W., Gehweiler, W. F., "CMOS/SOS Arithmetic Circuits," GOMAC Digest of Papers, June 1974, pp 74-75.

23. TRW Specification for MPY-16

24. Gaskill, J. R., Flint, J. H., Meyer, R. G., Micheel, L. J., and Weill, L. R., "LSI Multiplier Using High Speed ULG", IEEE Journal of Solid-State Circuits, Vol. SC-11, No. 4, Aug. 1976, pp 539-544.

25. Klose, D., Baird, T., Hampel, D., Rothweiler, J., "An LSI FFT Signal Detection and Demodulation Processor," (To be published in the GOMAC Proceedings), Nov. 1976.

26. Ali, Z., Macina, N. A., Rothweiler, J., "An FFT Processor For a 64-Tone MFSK LPI Reciever," RCA Technical Report SMDC-76-TR-001.

27. Macina, N. A., Nossen, S. J., Prost, K. J., "A Programmable Digital Quadrature Demodulator," RCA Technical Report SMDC-76-TR-003.

☆ U. S. GOVERNMENT PRINTING OFFICE: 1977 — 757-001/503